STOCHASTIC ESTIMATION AND CONTROL OF QUEUES WITHIN A COMPUTER NETWORK

Thesis

Mingook Kim, CAPTAIN, ROKA

AFIT/GCS/ENG/09-04

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/09-04

STOCHASTIC ESTIMATION AND CONTROL OF QUEUES WITHIN A COMPUTER NETWORK

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Mingook Kim

Captain, ROKA

March 2009

AFIT/GCS/ENG/09-04

STOCHASTIC ESTIMATION AND CONTROL OF QUEUES WITHIN A COMPUTER NETWORK

Mingook Kim

Captain, ROKA

Approved:

// signed //	4 March 2009
Lt Col. Stuart H. Kurkowski, Thesis advisor	date
// signed //	4 March 2009
Dr. Kenneth Hopkinson , Committee member	date
// signed //	4 March 2009
Capt. Ryan W. Thomas, Committee member	date

AFIT/GCS/ENG/09-04

Abstract

Captain Nathan C. Stuckey implemented the idea of the stochastic estimation and control for network in OPNET simulator. He used extended Kalman filter to estimate packet size and packet arrival rate of network queue to regulate queue size. To validate stochastic theory, network estimator and controller is designed by OPNET model. These models validated the transient queue behavior in OPNET and work of Kalman filter by predicting the queue size and arrival rate. However, it was not enough to verify a theory by experiment. So, it needed to validate the stochastic control theory with other tools to get high validity. Our goal was to make a new model to validate Stuckey's simulation. For this validation, NS-2 was studied and modified the Kalman filter to cooperate with MATLAB. Moreover, NS-2 model was designed to predict network characteristics of queue size with different scenarios and traffic types. Through these NS-2 models, the performance of the network state estimator and network queue controller was investigated and shown to provide high validity for Stuckey's simulations.

Table of Contents

List of Figures

List of Tables

STOCHASTIC ESTIMATION AND CONTROL OF QUEUES WITHIN A COMPUTER NETWORK

I. Introduction

In the future battlefield, military commanders and soldiers no longer have to yell coordinates over the radio and then plot maps to pinpoint their exact location. Instead, the Army is equipped with computer-driven networks on the battlefield that automatically know their locations at any time. The future battlefield aims to link soldiers to a wide range of weapons, sensors, and information systems by means of a mobile ad hoc network architecture that will enable unprecedented levels of joint interoperability, shared situational awareness, and the ability to execute highly synchronized mission operations. This means that the paradigms of the battlefield are rapidly changing and we need the new technologies to support these changes.

There are many causes changing the paradigms of future battlefield, but the key foundation is the network on a battlefield. This network is a main element to bring the future not only to the military but to normal society as well. In normal society, people use private mobile phone that allows them to communication with others and transfer Internet data. In the military, network system integrates sensors, weapons, command posts, command centers, and forward observers. Thus, the importance of network is huge regardless of the field.

The current routing and congestion control algorithms may not be sufficient for the future network. The main reason is the flexibility of the network. The current routing and congestion control algorithms do not support future flexibility. Flexibility was not important in times past because the network was the static topology with connected wires and less elements. Future networks will have dynamic topology and numerous elements. This will be because the network will not only require communication at organizational levels, but also a communication system between organizational levels, individual levels, and combinations of assets (i.e., in the military, sensors, effectors, command & control). In these cases, we will need dynamic algorithms to support the flexibility of the network.

Captain Nathan C. Stuckey proposes one solution to support dynamic networks [15]. He tries to improve the process of information delivery on computer networks to modify current routing and congestion control algorithms. For this improvement, he grafts stochastic nature onto the network algorithm. He does this because the current mainstream routing and congestion control algorithms do not use the probabilistic nature of computer networks. This means observed measurements cannot instantly influence the performance of the network. These measurements may contain a certain error, although it is impossible to know the current error of the network and fix it. However, if the stochastic nature of a computer network can be modeled accurately, the current state of the network can be estimated and the future state can be predicted. Being able to estimate the current and future state of the network would allow for improved routing and congestion control algorithms to be developed [15]. Stuckey designed the stochastic algorithm using Kalman filters to estimate the state of the network and develop optimal network control algorithms.

With this developed stochastic algorithm, Stuckey estimated current network state and control queue size to improve a performance of the network. Queues are one of the key elements in computer networks. In general, computer networks can be thought of as queuing networks. To demonstrate the possibility of applying stochastic control theory to computer networks, Stuckey developed a controller in his thesis to regulate queue sizes by controlling the packet arrival rate to the network queues. He designed network models to validate his algorithms simulated with OPNET and validated the process of applying stochastic control theory [15].

The goal of our research is to improve the process of the computer network. Following Stuckey, stochastic control theory was applied to estimate current network state and control queue size. With this stochastic control theory, we designed a network model to validate the performance of applying stochastic control theory under various network situations. Stuckey validated the process of the stochastic control algorithm, but it was insufficient to validate the performances of applying stochastic control theory to a real network. Although the OPNET tool is a powerful network simulation tool, it is necessary to validate the stochastic control theory with other tools to get high validity.

The main efforts of this research are to port it to another simulation tool and have more validity of stochastic control theory under various network situations. For this research, the network model developed with OPNET by Stuckey is ported to the Network Simulator-2 (NS-2) simulation tool [3]. And NS-2 is a network simulator developed at UC Berkely and is a well known, free, powerful network simulator tool. As will be more discussed fully in Chapter III, to demonstrate the performance of the stochastic control theory, we used various types of traffic sources. Also we tested the performance under various packet sizes, traffic rates, and prediction times.

The following chapter will provide more detail on the background of these aspects and ideas used in this research. Chapter III discusses the methods for validating the OPNET model developed by Stuckey with the NS-2. Chapter III also discusses the various network situations used to validate the performance of applying stochastic control theory. Chapter IV provides results and analyses of the data collected by the NS-2 model to validate the theory of Stuckey's work, as well as validate the work of the NS-2 model that was ported. Finally, Chapter V concludes the research by stating whether or not the objectives of the research have been accomplished, and provides recommendations for possible future research in this area of study.

II. Literature Review

2.1. Introduction

While the current algorithms in use today may be sufficient for static networks, in the future we need active control algorithms. For these active algorithms, we need to know the state of the network such as the path information and the rate over the network. Advanced control algorithms require knowledge of the various network states. So in this chapter, we introduce how to determine the state of the network.

2.2. Network Tomography

Network tomography is the research of a network's internal characteristics using information derived from network states. The word “tomography” is used to emerging the field, to other processes that infer the internal characteristics of an object from external measurement, as is done in X-rays imaging.

2.2.1. Network Tomography Basics

Today’s significant network problems can be classified according to the acquisition type of network information and the performance parameters of interest [20]. That is, we can discuss broadly that the network’s problem of inferring the network state involves estimating network performance based on traffic measurements at a limited and designed subnet of the nodes.

However, the real network problem is more complex. Sometimes we cannot measure the performance of a real network model or to discover network problems, because the characteristic of computer networks are complex and decentralized.

Moreover, the hardware for the network or computer is developed quickly and mobile or wireless hardware makes it difficult to measure network states. This means that in the past, the network problems were simple processes of arithmetic but today these are the processes of estimation and prediction.

Y. Vardi studied these types of problems, and became one of the first researchers to coin the term network "tomography" [20]. The network tomography is derived from the similarity between network and medical tomography. Recently, two forms of network tomography have been addressed in the literature: i) link-level parameter estimation based on end-to-end, path-level traffic measurements and ii) sender-receiver path-level traffic intensity estimation based on link-level traffic measurements [20]. In link-level parameter estimation, the traffic measurements typically consist of counting packets transmitted and/or received between nodes or time delays between packet transmissions and receptions. In path-level traffic intensity estimation, the measurements consist of counting packets that pass through nodes in the network.

2.2.2. Network Tomography Application

As presented in the basics of network tomography, most network measurement schemes have existed on the outside edges of the network. In link-level or path-level, these edges are the same concept, and this means transmitted and received nodes. Of course, the states of the network must be inferred from the information obtained at the edges.

Once measurements are obtained from the edges, statistical inference techniques are used to analyze the measurement data in order to learn as much as possible about the state of the network. Numerous statistical techniques have been used including complexity-reducing hierarchical statistical models, moment- and likelihood-based estimation, expectation- maximization and Markov chain Monte Carlo algorithms [20].

One statistical technique that has had only limited use in the network tomography field is the Kalman filter. The Kalman filter is fed measurements from the system of interest and produces an estimate of the system state. Within the Kalman filter, the system is modeled as a set of differential equations in the continuous-time case or as a set of difference equations in the case of a discrete-time system. The system model is used to propagate the estimate of the state of the system forward in time until a measurement is received. At this point, the system state attained from the measurement is compared to the estimate of the system state and combined in an optimal manner.

2.3. Kalman filter

Forecasts are rarely perfect, as they show what is likely to happen “on average”. So it is a good practice to complement forecasts with measures of the forecast uncertainty. The most common measure of uncertainty is the variance. The Kalman filter is an iterative computational algorithm designed to calculate forecasts and forecast variances for some models.

2.3.1. Kalman Filter Background

In 1960, R.E. Kalman designed a recursive solution to the discrete-data linear filtering problem. Since that time, due in large part to advances in digital computing, the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation [4].

The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error (see Section 2.6.1). The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown.

The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of measurements. As such, the equations for the Kalman filter fall into two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the a priori estimates for the next time step. The measurement update equations are responsible for the feedback—i.e. for incorporating a new measurement into the a priori estimate to obtain an improved a posteriori estimate.

The time update equations can also be thought of as predictor equations, while the measurement update equations can be thought of as predictor equations. Indeed the final estimation algorithm resembles that of a predictor-corrector algorithm for solving numerical problems as shown below in Figure 1.

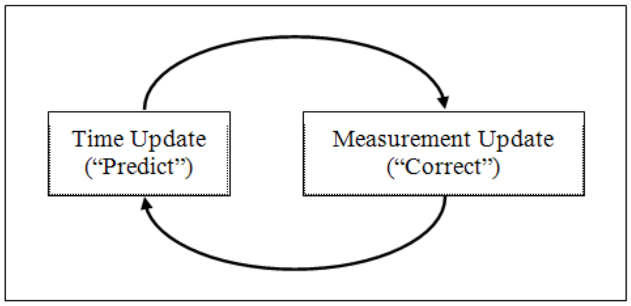


Figure 1. The ongoing discrete Kalman filter cycle [4]

2.3.2. Application of Kalman Filter

The Kalman filter is a multi-input, multi-output, recursive digital filter that can optimally estimate, in real time, the states of the system based on its noisy outputs. The estimates are statistically optimal in the sense that they minimize the mean square estimation error. As a result, the Kalman filter has been used to estimate the optimal solution for various real applications.

This filter has applications in GPS based navigation systems like Wide Area Augmentation System [1]. Several production GPS receivers use the Kalman filter [16]. A GPS receiver, however, doesn't measure exact range to each satellite. The measurement process is corrupted by noise which introduces errors into the calculation. A Kalman filter characterizes the noise sources in order to minimize their effect on the desired receiver outputs. When the GPS receiver is aided or integrated with other navigation sensors (e.g., Inertial Navigation Systems (INS) [21], clock, altimeter or Attitude Heading Reference System (AHRS) [2]), then the Kalman filter can be extended to include the measurements added by these sensors. In fact, a typical implementation for integrated systems would be to have a central Kalman filter incorporating measurements from all available sources [16].

Some universities use Kalman filters to play robot soccer [19]. Here, robot localization is the process whereby a robot determines its own position in the world in which it functions with a Kalman filter. World modeling is the process used by a robot to determine the position of other objects in its world. The Kalman filter has been implemented for a robot to locate its own position and the ball position.

An application of the Kalman filter has been developed for the real-time identification of a distributed parameter in a nuclear power plant. This technique can be used to improve numerical method-based best-estimate simulation of such a complex system [10].

The Kalman filter can use one solution to minimize network traffic between mobile users. When the devices in the GRID wireless are highly mobile, there will be much traffic required to exchange the geographic position information of each mobile node, and this causes an adverse effect on efficient battery usage. Each node uses the algorithm to estimate its own movement (also other node's movement), and when the estimation error is over a threshold, the node sends the UPDATE (including position, velocity, etc) packet to other devices. As the estimation accuracy is increased, each node can minimize the number of UPDATE packet transmission [22]. Also, the Kalman filter based prediction algorithm makes significant contributions to various aspects of network traffic engineering and resource allocation. The prediction algorithm with Kalman filter yields superior prediction accuracy than the other algorithms [9].

As network data is transmitted in randomly sized packets, the Kalman filter can be used to estimate the optimal packet size in a wireless network in order to maximize channel efficiency [23].

2.4. Network Queue

Network queueing is a very important application of queueing theory. A queue is a waiting line that is like a sequence of messages or jobs held in temporary storage awaiting transmission or processing. In the network system, the queue is same meaning but the objective is different from other application. Figure2 shows simple queueing model that the simple queueing model involves a single queue.

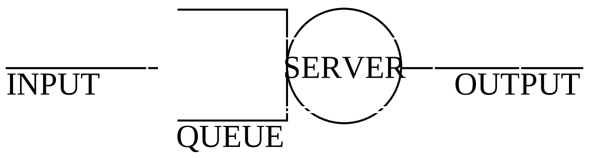


Figure 2. The ongoing discrete Kalman filter cycle [27].

In this queueing system, inputs – messages or packets requesting service – come from the left and wait for service in the queue. When the server can serve the request from the input, it services request from queue and then sends service or output to the right.

Although this model is simplest queueing system model, in general, the aim of traffic modeling is to provide the network designer with relatively simple methods to characterize traffic load on a network. Ideally, such means can be used to estimate performance and to enable efficient provisioning of network resources. Thus, we can know understand the importance of queue with this simple model. Actually, in the network, queue size would impact the performance of network. In real application, queue size is limited and queue size is able to influence delay and throughput.

As a result, in dynamic routing and congestion control system based on stochastic algorithm that is developed for the future, Queueing system is key point of network and that is reason why we manage queue size with Kalman filter to improve performance of network.

2.5. Traffic Type

Modeling a traffic stream emitted from a source, or a traffic streams that represents a multiplexing of many Internet traffic streams, is part of traffic modeling.

These traffic streams have an each characteristic that is called as traffic type. In this research, we chose several types of network traffic like Constant Bit Rate (CBR), Exponential, Pareto and Poisson. These traffic models loosely represent real network data flow. And these traffic models are defined by TrafficGenerator under the NS-2 [25].

2.5.1. CBR Traffic

This traffic model generates traffic according to a deterministic rate. Packets are of constant size. Optionally, some randomizing dither can be enabled on the interpacket departure intervals. A CBR object is embodied in the OTcl class Application/Traffic/CBR. The member variables that parameterize this object are shown in Table1.

Table 1. The configuration variables of CBR traffic.

Name	Description
rate_	the sending rate
interval_	interval between packets
packetSize_	the constant size of the packets generated
random_	flag indicating whether or not to introduce random “noise” in the scheduled departure times
maxpkts_	the maximum number of packets to send

2.5.2. Exponential Traffic

This traffic model generates traffic according to an Exponential distribution. Packets are sent at a fixed rate during “on” periods, and no packets are sent during “off” periods. Both “on” and “off” periods are taken from an exponential distribution. Packets are constant size. An Exponential On/Off object is embodied in the OTcl class

Application/Traffic/Exponential. The member variables that parameterize this object are shown in Table2.

Table 2. The configuration variables of Exponential traffic.

Name	Description
packetSize_	the constant size of the packets generated
burst_time_	the average “on” time for the generator
idle_time_	the average “off” time for the generator
rate_	the sending rate during “on” times

2.5.3. Pareto Traffic

This traffic model generates traffic according to a Pareto On/Off distribution. This is identical to the Exponential distribution, except the on and off periods are taken from a Pareto distribution. These sources can be used to generate aggregate traffic that exhibits long range dependency like a Web traffic generator assuming that Web object sizes form a Pareto distribution. A Pareto On/Off object is embodied in the OTcl class Application/Traffic/Pareto. The member variables that parameterize this object are shown in Table3.

Table 3. The configuration variables of Pareto traffic.

Name	Description
packetSize_	the constant size of the packets generated
burst_time_	the average "on" time for the generator
idle_time_	the average "off" time for the generator
rate_	the sending rate during "on" times
shape_	the "shape" parameter used by the Pareto distribution

2.5.4. Poisson Traffic

A Poisson object is instantiated by the OTcl class Application/Traffic/Poisson. Packet generation is a Poisson process: the time between successive packets is independently generated from an exponential distribution with mean interval_. The important member variables that parameterize this object are shown in Table4.

Table 4. The configuration variables of Poisson traffic.

Name	Description
rate_	the sending rate
interval_	mean packet inter-generation time
packetSize_	size of the packet generated
maxpkts_	maximum number of packets to send

2.6. Statistical Simulation Methods

In this thesis, we need some method for qualitative assessment of the model. Some output data have variable qualitative measures; we also need rules to compare each output data with verifiability. Therefore this section we discuss methods that are satisfied with above conditions.

2.6.1. Mean Squared Error

Mean squared error (MSE) is a classical deviance measure for absolute scale [24]. We assume y_t to be the measurement of a typically time continuous real world process and $\hat{y}_t$ simulated values of the same process at the same times $t_1, \dots . t_n$. we only consider real valued y_t and not vectors. Concerning the methods of this section multiple outcomes can be studied individually. In this situation, mean squared error measure is defined by:

$$\textbf{mean squared error}: \mathbf{MSE} = \frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2 \qquad (2.\ 1)$$

MSE measures absolutely, based on scale dependent measures of model performance, i.e. MSE can only provide a relative comparison between different models. These deviance measures are zero if and only if the values are identical. Table5 shows how the mean square error would be calculated. Then it would add up the square errors and take the sum. In this case the sum of the errors is 43.1375 and the mean square error is 6.1625. We can use this to compare actual queue sizes with the Kalman filter predicted levels.

Table 5. Example of mean squared error (MSE) [11].

Time (t)	Weight (Y)	Estimated ($\hat{Y}$)	$(Y - \hat{Y})^2$
1	42	36.53	29.9209
2	46	45.77	0.0529
3	51	51.93	0.8649
4	57	58.09	1.1881
5	62	64.25	5.0625
6	68	70.41	5.8081
7	73	73.49	0.2401

2.6.2. Mean Absolute Percentage Error

Mean absolute percentage error (MAPE) is a deviance measurement for relative scale [24]. Mean absolute percentage error is the average value of the absolute values of errors expressed in percentage terms. We consider data to be in a relative scale if they are strictly positive and the importance of the difference is given by the ratio and not by the arithmetic. In the same situation as section 2.6.1, mean absolute percentage error is defined by:

$$\textbf{mean absolute percentage error} : \mathbf{MAPE} = \frac{100}{n}\sum_{t=1}^{n}\frac{|y_t-\hat{y}_t|}{|y_t|} \tag{2. 2}$$

MAPE cannot be determined if measured values y_t are equal to zero and it tends to infinity if measurements are small or near to zero. This is a typical behavior, when relative errors are considered. Table6 shows how the mean absolute percentage error would be calculated. We can also use this to compare actual queue sizes with the Kalman filter predicted levels.

Table 6. Example of mean absolute percentage error (MAPE).

Time (t)	Weight (Y)	Estimated ($\hat{Y}$)	$\frac{\lvert Y-\hat{Y}\rvert}{\lvert Y\rvert}$
1	42	36.53	0.1302
2	46	45.77	0.005
3	51	51.93	0.0182
4	57	58.09	0.0191
5	62	64.25	0.0363
6	68	70.41	0.0351
7	73	73.49	0.0067

2.6.3. Monte Carlo Method

Monte Carlo method provides approximate solutions to a variety of mathematical problems by performing statistical sampling experiments. They can be loosely defined as statistical simulation methods, where statistical simulation is defined in quite general terms to be any method that utilizes sequences of random numbers to perform the simulation [8]. Thus Monte Carlo methods are a collection of different methods that all basically perform the same process. This process involves performing many simulations using random numbers and probability to get an approximation of the answer to the problem.

The Monte Carlo method is just one of many methods for analyzing uncertainty propagation, where the goal is to determine how random variation, lack of knowledge, or error affects the sensitivity, performance, or reliability of the system that is being modeled [12]. Monte Carlo simulation is categorized as a sampling method because the inputs are randomly generated from probability distributions to simulate the process of sampling from an actual population. So, we try to choose a distribution for the inputs that most closely matches data we already have, or best represents our current state of knowledge. The data generated from the simulation can be represented as probability distributions (or histograms) or converted to error bars, reliability predictions, tolerance zones, and confidence intervals (as shown in Figure 3).

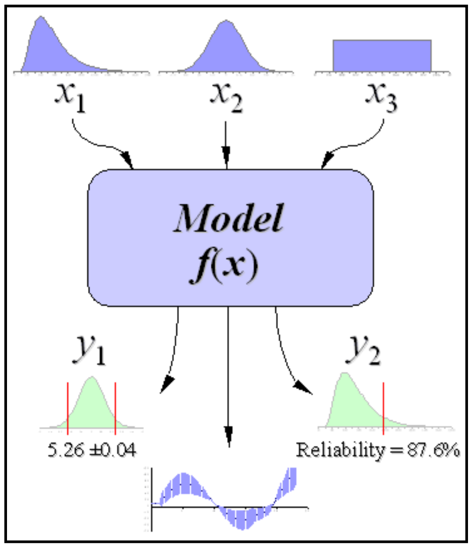


Figure 3. The basic principle behind Monte Carlo simulation [12].

The steps in Monte Carlo simulation corresponding to the uncertainty propagation shown in Figure 3 are fairly simple, and can be easily implemented in a spreadsheet like Excel for simple models. All we need to do is follow the five simple steps listed below:

Step 1: Create a parametric model, $y = f(x_1, x_2, \ldots, x_q)$.

Step 2: Generate a set of random inputs, $x_{i1}, x_{i2}, \ldots, x_{iq}$.

Step 3: Evaluate the model and store the results as y_i.

Step 4: Repeat steps 2 and 3 for i = 1 to n.

Step 5: Analyze the results using histograms, summary statistics, confidence intervals, etc.

We will simulate the network model to approximate the performance of the stochastic algorithm using Kalman filter with Monte Carlo method.

2.7. Review of Developed model for stochastic algorithm

This section discusses the OPNET and the NS-2 network simulation tool. Moreover, it reviews the network estimator model and control model developed by Stuckey.

2.7.1. OPNET

OPNET Modeler is the industry's leading simulator specialized for network research and development [17]. It allows a researcher to design and study communication networks, devices, protocols, and applications with great flexibility. It provides a graphical editor interface to build models for various network entities from physical layer modulator to application processes. All the components are modeled in an object-oriented approach which gives intuitive easy mapping to real systems. It gives a flexible platform to test your new ideas and solutions with low cost.

In Stuckey's thesis, his algorithm was validated for ideal traffic using a network simulated in OPNET. And then OPNET is used to design the transient queue model to test the adequacy of the transient queue size model [15]. It is obvious that the OPNET tool provides powerful network simulation to test modeling traffic and network topology, but it is not enough to validate the algorithm with just one network simulation tool. OPNET may or may not have a little difference between the model and the actuality. For that reason, it needs other network tool to model Stuckey's algorithm like the NS-2.

2.7.2. NS-2

NS-2 is the second version of a network simulator tool developed by the Virtual InterNetwork Testbed (VINT) project [3]. It is an event-driven network simulator, which is popular with the networking research community. It includes numerous models of common Internet protocols including several newer protocols, such as reliable multicast and TCP selective acknowledgement. A network animator, Nam, provides packet-level animation and protocol specific graphs for design and debugging of network protocols. Additionally, different levels of configuration are present in NS-2 due to its open source nature, including the capability of creating custom applications and protocols as well as modifying several parameters at different layers.

The NS-2 architecture takes an object-oriented approach using C++ and Otcl [26]. Otcl is an object-oriented variant of the well-known scripting language tcl. C++ is utilized for per-packet processing and forms the core of the simulator. Otcl is used for simulation scenario generation, periodic or triggered action generation and for manipulating existing C++ objects.

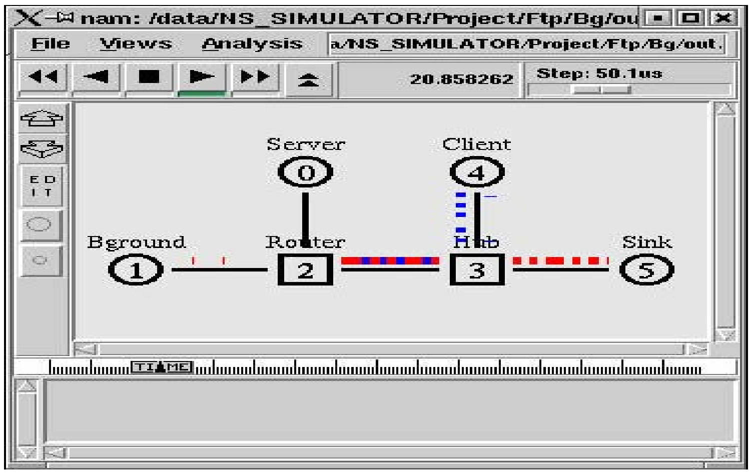


Figure 4. Network Representation in NS-2 [3].

Figure 4 shows a network representation with the NS-2 components visualized in NAM. Nam is a Tcl/TK based animation tool for viewing network simulation traces and real world packet trace data [25]. The Nam trace file should contain topology information like nodes, links, queues, node connectivity etc as well as packet trace information shown in Figure 4. The design theory behind Nam was to create an animator that is able to read large animation data sets and be extensible enough to understand invisible network situation.

2.7.3. Integrated Network Estimator

The Kalman filter designed by Stuckey was integrated into the OPNET simulations to demonstrate the operation of a network queue estimator in a computer network. Since the Kalman filter was implemented in MATLAB and the network simulation was implemented in OPNET, a co-simulation interface was developed between the two programs. OPNET was used as the primary program and maintained control of the simulations. When Kalman filter computations were needed, the MATLAB functions were called by the OPNET simulation. Application programming interface

(API) of C language that allows the MATLAB engine to be called by C code is included with the MATLAB software. Since OPNET models are constructed of C code, this API was used to call MATLAB from within the OPNET models.

The integrated network estimator model is shown in Figure 5. The integrated network estimator is designed with Kalman filter as described above. the router and Client 1 model are modified. Duplex point-to-point links are used to connect the nodes. The traffic was sent from Client 1 to Client 2 using the UDP transport protocol. Poisson traffic was sent with a mean arrival rate of five packets/second and a mean packet size of 1500 bytes. With this simple model, we can estimate the queue and control the queue with Stuckey's theory.

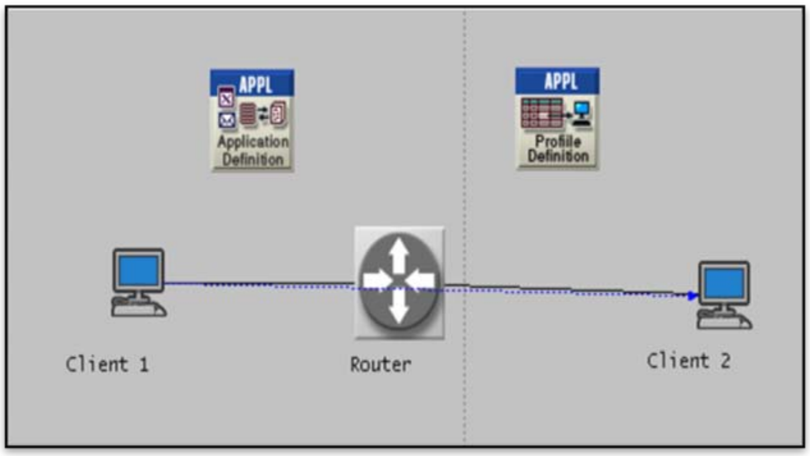


Figure 5. Integrated network estimator model from Stuckey's thesis [15].

The modified router measures current state of network and the modified client takes the network state from router and estimate the queue size. The following is more detail description about this integrated network estimator modeled by the OPNET.

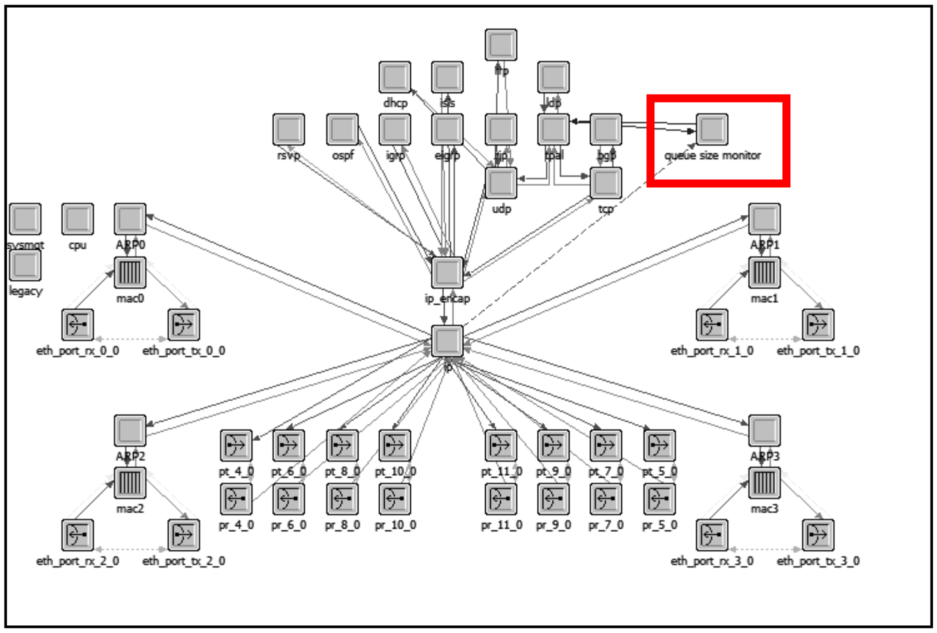


Figure 6. Modified router model [15].

The internal model of the router from figure 5 is shown in Figure 6. The IP block has been modified to observe the IP packets traveling through the router. The queue size monitor block has been added to the router model. The queue size monitor block contains the queue size monitor application. The function of the queue size monitor application is to send the router queue size data to all of the senders that are transmitting data through the queue. The workstations receive queue size data packets from the routers. This packet information is sent to the queue size monitor block. The size of the central queue in the router is also sent to the queue size monitor block from the IP block upper right corner of Figure 6.

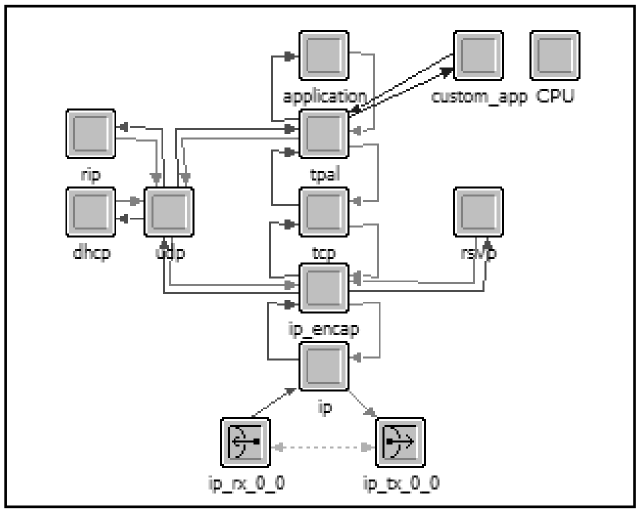


Figure 7. Modified workstation model [15].

The Kalman filter block has been added into the workstation model. The modified workstation model is shown in Figure 7. When the workstation receives a queue size data packet, the interface forwards it to the Kalman filter block. The Kalman filter block receives the queue size data packet and calls the Kalman filter MATLAB code that was validated through the co-simulation interface. The Kalman filter code processes the measurement data by executing the propagate and update steps. The estimate of queue size and packet arrival rate is then passed back to OPNET and recorded.

2.7.4. Integrated Network Controller

The simple OPNET model used to test the controller design is shown in Figure 8. The two dashed lines are statistic lines that report the queue size and queue arrival rate to the source. The source and queue models are modified versions of the standard OPNET models used in the integrated network estimator and the sink model remains unchanged. The queue model has been slightly modified to record the arrival rate to the queue. The source model has been modified so that the sending rate is no longer set by the user, but computed by the controller. Since the controller is implemented in MATLAB, the source interacts with the controller through the co-simulation interface.

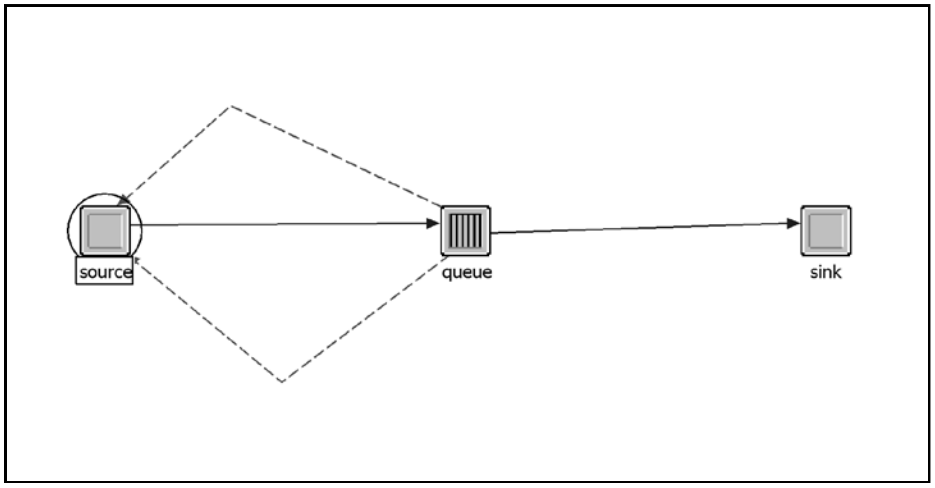


Figure 8. Simple controller model implemented with OPNET [15].

2.8. Summary

In this chapter, The Kalman filter was described with a few examples of the use of Kalman filter within the network tomography. And some issues for simulation were presented to help the analysis of the performances of Kalamn filter. Also, it reviewed the OPNET mdoel implemented by Stuckey and the NS-2 simulation tools. In the following chapter, we discuss how network state estimator and controller are validated under the different traffic type and scenarios.

III. Methodology

3.1. Introduction

Our stochastic control scheme is designed with a Kalman filter to estimate network state and to regulate queue size within a dynamic computer network. For this stochastic control scheme, Stuckey made a simulation model that are presented for both the Kalman filter based network state estimator and the network queue controller [15]. He designed a network state estimator and network queue controller model with OPNET and implemented an extended Kalman filter with MATLAB (see Chapter II for more details). In this chapter, we show other tool and models to validate his stochastic control scheme and use of his models.

3.1.1. Goals

The first goal of this research is to develop and validate the stochastic control scheme with the NS-2 simulation tool. Stuckey used OPNET simulation to validate his theory. But this research use the NS-2 to call the function of Kalman filter that was written in MATLAB. The second goal was to predict network characteristics of queue size with different scenarios and traffic types.

3.1.2. Approach

With collaboration from Stuckey, we made the modifications to the Kalman Filter. Instead of looking at past information and propagating to the present, the present data is used to assess the future. And as mentioned previously, we port and compile the code into NS-2. The Kalman Filter was written in MATLAB and executed through the

C/C++ MATLAB engine. NS-2 is also written in C++ with scripting support in TCL. We created two new C++ files for the port: the first being the code that would call on the modified Kalman Filter and second a drop tail queue that would send state information to the filter. This attempt worked after linking the MATLAB libraries with the NS-2. Finally, we needed to add the framework to call the filters simultaneously. Another C++ file was added and incorporated with the TCL scenario script to act as a timer. The timer will fire every “x” seconds calling all the filters in the network to execute the next prediction.

3.2. Setup of NS-2 and MATLAB

To link the MATLAB libraries with the NS-2, we needed to setup the configuration of the NS-2 and modify some files of system. We also modified MATLAB files detailed below.

3.2.1. Modify .bashrc file

This configuration file provides user-definitions to command aliases and PATH for that user's bash shell. To run simulation, some library files and Tcl files have difference places. So we need to make a few aliases and some PATH modifications. This help that create an executable file for linking with .cpp/ .cc to see the Makefile.in additions.

```
MATLAB= /*matlab path here*/
export MATLAB
ARCH=glnxa64
export ARCH
LD_LIBRARY_PATH=$MATLAB/bin/$ARCH:
$MATLAB/sys/os/$ARCH:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

Figure 9. Modified .bashrc file.

3.2.2. Add MATLAB directories

This part describes how the MATLAB directory is setup which can be placed right into the NS-2.29 directory. This MATLAB directory is a copy directory from original MATLAB files.

```
matlab/bin:  This directory contains MATLAB library files as well as the
             c/matlab compiliers.  All these can be found under
             $MATLAB/bin.
     bin/engopts.sh:  The c/matlab compiler to link the MATLAB and c
                      libraries
     bin/mex:  The mex compiler
     bin/glnxa64:  Contains all the underlying MATLAB library files (.so)
matlab/extern:  This directory contains the MATLAB include files.  All these
                files can be found under $MATLAB/extern.
     extern/include:  Contains all the header files needed to run the
                      MATLAB engine.
```

Figure 10. MATLAB directories to add into NS-2.

3.2.3. Modify Makefile.in

This part describes the additions that are needed in the Makefile.in for NS-2.29 source. After changing the file, a "./configure" will need to be performed to re-configure the new Makefile. Once this has been completed the "make" command will recompile all the code and link the MATLAB libraries. The modified sections of the MAkefile.in will go through to the Makefile to update.

```
            # The MATLAB include folder as well as the MATLAB directory have .h files that
need to be included

                              INCLUDES = \
                                   -I. \
                                   @V_INCLUDES@ \
                                   -I./wpan \
                                   -I./matlab/extern/include -I./matlab \

            # The MATLAB library files are in the bin/glnxa64.  In order to link the libraries the
"–lm @LIBS@ -lmx -lmex – lmat –leng" line is needed.  This will link with the MATLAB
engine, matrix, and Mex libraries.

                                   LIB     = \
                                           @V_LIBS@ \
                                           @V_LIB@ \
                                           @V_LIB@ -L./matlab/bin/glnxa64 \
                                           -lm @LIBS@ -lmx -lmex -lmat -leng
                                   #       -L@libdir@ \

            # The object files need to be under the OBJ_CC section.  We have added kalmanFilter,
kalmanFilter-drop-tail, and KFTimer

                        OBJ_CC = \
                             common/scheduler.o common/object.o common/packet.o \
                             wpan/p802_15_4trace.o wpan/p802_15_4transac.o \
                             matlab/kalmanFilter.o queue/kalmanFilter-drop-tail.o \
                             queue/KFTimer.o \

            @V_STLOBJ@
```

Figure 11. Modified Makefile.in.

3.3. Implement ported into NS-2

As described in section 3.1.2, KFTimer class incorporated with the TCL scenario script to act as a timer. The function predictionLoop() in this class call the function updatePrediction() from kalmanFilter-drop-tail for every "x" prediction times to execute the next prediction. The outline of the flow of the function is following.

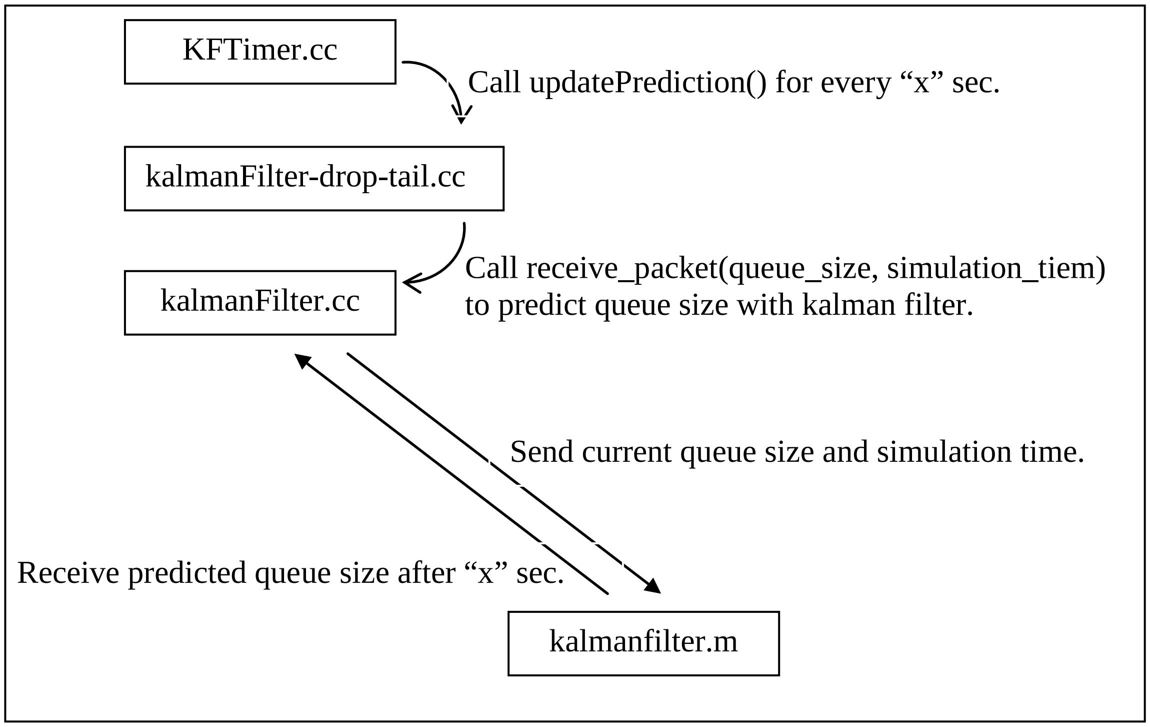

KFTimer.cc
Call updatePrediction() for every "x" sec.
kalmanFilter-drop-tail.cc
Call receive_packet(queue_size, simulation_tiem)
to predict queue size with kalman filter.
kalmanFilter.cc
Send current queue size and simulation time.
Receive predicted queue size after "x" sec.
kalmanfilter.m

Figure 12. Flow of function and file implemented by NS-2.

Shown in Figure 12, kalmanFilter-drop-tail.cc make kalmanFilterDropTail class that operates like DropTail queue. This class defined the activity of queue like input of queue or output of queue, but in this modified kalmanFilterDropTail queue is added function to send network information to kalmanFilter class. The KalmanFilter class makes memory space for MATLAB variables, initialize the variables in MATLAB and send current queue size and simulation time to kalmanfilter.m that is MATLAB file implemented by Stuckey. Kalmanfilter.m calculates the future queue size with Kalman filter theory and reply to kalmanFilter class.

3.3.1. KFTimer Class

This section describes more detail KFTimer class. The following figure is the detail description about C++ file was added and incorporated with the TCL scenario script to act as a timer.

```
/*  KFTimer.cc
 *  This is the KFTimer class that will be instantiated in the tcl script to
 *  act as an internal timer to call the kalman filter MATLAB code.
 */
#include "KFTimer.h"

//Constructor:
static class KFTimerClass : public TclClass {
public:
       KFTimerClass() : TclClass("KFTimer") {}
  TclObject* create(int, const char*const*) {
    return (new KFTimer());
  }
} class_KFTimer;

/* Method to interface with tcl through command line.  After instantiating
 * the KFTimer in the tcl we can perform this method.
 *
 *  "predict" = call the predict loop method to run all the MATLAB
 *              engines for each of the queues
 */
int KFTimer::command(int argc, const char*const* argv)
{
       if (argc==2) {
              if (strcmp(argv[15], "predict") == 0) {
                     predictLoop();
                     return (TCL_OK);
              }
       }
       return Connector::command(argc, argv);
}

/* Loop through the kalman filter drop tail queues to perform the update
 * prediction method which calls the MATLAB engine
 */
void KFTimer::predictLoop(){
       for(int i = 0; i<(int)qList.size(); i++){
              qList[i]->updatePrediction();
       }
}
```

Figure 13. The code of KFTimer Class.

3.3.2. KalmanFilter-drop-tail Class

KalmanFilter-drop-tail class is almost same as the drop tail queue. Like drop tail queue, this KalmanFilter-drop-tail queue drop the packet when the queue would overflow. The following figure is the modified part from the drop tail queue.

```
/*
 *  KalmanFilter-drop-tail.cc
 *  This is the KalmanFilter-drop-tail class that defined the act of queue.
 *  And by the KFTimer class, this queue send the network information to
 *  Kalman filter class
 */

void KalmanFilterDropTail::updatePrediction(){

        kf->receive_packet(q_->length(), Scheduler::instance().clock());

}
```

Figure 14. The function updatePrediction() of KalmanFilter-drop-tail Class.

Every “x” second, the KFTimer class calls the function updatePrediction() of the KalmanFilter-drop-tail class. The function updatePrediction() call the function receive_packet() of the kalmanFilter class. When the kalmanFilter class is called, the current queue size and simulation time is sent as the network information to kalmanfilter class

3.3.3. KalmanFilter Class

KalmanFilter class is key class to incorporate NS-2 and MATLAB. In this class, the many variables are ready and are initialized for the MATLAB. The following figure is the detailed description about kalmanFilter class calling the MATLAB function.

```
/*
 *  kalmanFilter.cc
 *  This is the Kalman Filter class that will run the MATLAB engine to
 *  performs Nathan Stuckeys's extend Kalman filter implementation.
 */

/* Method to initialize the variables in MATLAB.  init_kalman_filter needs to
 * be called before we can receive a network information.
 */
void kalmanFilter::init_kalman_filter(){

     /***** omission *****/

/* The path below needs to be where the kalmanfilter.m file is located.
 */
      engEvalString(ep, "cd('/home/afiten3/gcs09m/mkim/ns-allinone-2.29/ns-
2.29/matlab/workspace')");
}

/**
 * Method should be called everytime kalmanFilter-drop-tail class call.
 * Queue size and simulation time are inputs from the kalmanFilter-drop-tail
 * class.
 */
void kalmanFilter::receive_packet(double queueSize, double sample_time){
             double meas_data[2*1];
             double* output_data_ptr;
             double output_data[1*5];
             mxArray* output;

             // Open the file to output the data with the unique file name
             data.open(os.str().c_str(), ios::app);

             // Ouput results to a file
             meas_data[0] = queueSize;
             meas_data[15] = sample_time;
             memcpy(meas_ptr, meas_data, 2*1*sizeof(double));
             engPutVariable(ep, "data", meas);

             engEvalString(ep, "kalmanfilter");

             output = engGetVariable(ep, "output");
             output_data_ptr = mxGetPr(output);

             memcpy(output_data, output_data_ptr, 1*3*sizeof(double));

             data << sample_time << "\t" << queueSize << "\t"  <<
output_data[15] << "\t" << output_data[0] << "\n";
             mxDestroyArray(output);

             data.close();
}
```

Figure 15. The key functions of kalmanFilter Class.

This kalmanFilter class receives the output data (predicted queue size) from MATLAB function and makes unique output txt file. In this output txt file, it stores by the order as current time, current queue size, future time and future queue size. Therefore we can know the performance of Stuckey's theory by comparing the current queue size and predicted queue size in this txt file.

3.4. Validation

To validate the model of NS-2 like queues and links provided, we need to make a simple queue model that consisted only of the NS-2 network object. The model will verify under ideal conditions that exactly match the input traffic assumptions made during the development of the Stuckey's model. Then, the network queueing model will be verified by a more detailed network simulation that involves complex models of network hardware operating under ideal traffic situations.

Next, to check that NS-2 code can call the MATLAB engine, we made a test function. This function just prints a simple message that is received from the MATLAB code. We know that communication occurred between NS-2 and MATLAB as soon as we see the message printed to the screen. So if we have some problem with NS-2 and MATLAB, we can see error message.

By testing operation between NS-2 and MATLAB with above method, we can verify the communication between NS-2 and MATLAB but, this cannot verify the result from the calculation that NS-2 call the MATLAB engine to predict the queue size. So, we will make other input data that is network information sent to MATLAB engine by the NS-2, and calculate the predicted queue size with MATLAB program and the MATLAB function that is implemented by the Kalman filter. Finally, we can verify to compare the result made by communication between NS-2 and MATLAB, and the result calculated by only MATLAB program.

Last, the formula of the extended Kalman filter does not need to verify the exactness because we use the same MATLAB file implemented by Stuckey. The formula is verified by the Stuckey's thesis [15].

3.5. Evaluation

To validate the stochastic control scheme, we make multiple computer network scenarios. To validate Kalman filter that is called by NS-2 with MATLAB, we make simple network, has simple nodes to estimate network state and to regulate queue size. We run these models to validate with multiple types of network traffics and varying time to add verification.

3.5.1. Type of Traffic

In this research, we chose several types of network traffic like Constant bit rate (CBR), Exponential, Pareto and Poisson. These traffic models represent real network data flow. And these traffic models are defined by TrafficGenerator with NS-2.

3.5.1.1. CBR Traffic

In NS-2, the CBR traffic generator generates traffic according to a deterministic rate. Packets are constant size. In ATM networks, voice traffic is usually represented as CBR traffic [7]. To verify under more network situation, we will vary the configuration of CBR traffic. In CBR traffic, we will derive two situations from the queue. The first situation is the utilization (ρ) is more than one and the queue size will increase infinitely.

Another situation is the utilization (ρ) is less than one and the queue size will stay under the queue limit. Basically the sending rate is five packets/second and service rate is 5.88 packets/second. This is chosen as a reasonable rate and used in Stuckey's thesis.

As described in Section 2.5.1, CBR traffic generator has five configuration variables of CBR traffic. But in this simulation, we will change only two configuration variables: sending rate and packet size. This is because other configuration variables do not effect this simulation or are mutually exclusive from these two variables. Table 7 shows the detail specification for testing the simulation.

Table 7. The configuration variables of CBR traffic.

Name	Default	Setting Value	Test Range
rate_	448 Kbit/sec	7500 bits/sec	7300,7400,7500,7600, 7700 bits/sec
interval_	1 sec	Default	Default
packetSize_	210 bit	1500 bits	1300,1400,1500,1600,1700 bits/sec
random_	Off	Off	Off
maxpkts_	$2^{28}\ packets$	Default	Default

As described above, packet size and arrival rate is setting for the utilization is less than one.

3.5.1.2. Exponential Traffic

Exponential traffic generator generates traffic according to an Exponential distribution. Packets are sent at a fixed rate during “on” periods, and no packets are sent during “off” periods. Both “on” and “off” periods are taken from an exponential distribution. Basically, we keep the utilization (ρ) less than one. This is because the average number of queue size will increase infinitely as $\rho \rightarrow$ one. Therefore we choose a utilization of 0.85 to load the queue sufficiently in order to provide a reasonable response. Like above section CBR, sending rate is five packets/second and service rate is 5.88 packets/second as Stuckey's thesis.

As described in Section 2.5.2, Exponential traffic generator has four configuration variables however in this simulation, we will change this two configuration variables and measure the influence of this changing. Burst time and idle time is the average time about "On/Off" for generating the packets. So we will keep the default value for this burst time and idle time. The following Table 8 is the detail specification for testing simulation.

Table 8. The configuration variables of Exponential traffic.

Name	Default	Setting Value	Test Range
packetSize_	210 bit	1500 bits	1300,1400,1500,1600,1700 bits/sec
burst_time_	0.5 sec	0.7 sec	Not change
idle_time_	0.5 sec	0.3 sec	Not change
rate_	64 Kbit/sec	7500 bits/sec	7300,7400,7500,7600, 7700 bits/sec

3.5.1.3. Pareto Traffic

Pareto traffic generator can be used to generate traffic like Web traffic. This generator like Exponential traffic sends packets at a fixed rate during "on" periods, and no packets are sent during "off" periods. Therefore, like Exponential traffic's configuration, we will choose a utilization of 0.85, sending rate is five packets/second and service rate is 5.88 packets/second. Also, burst time and idle time will use default values. The Pareto traffic generator has specific parameter "shape" that influences the Pareto distribution. But we use default values for "shape", because we will test the same simulation as other traffic generator and "shape" parameter does not influence the simulation performance if we use the same value for all tests. We will change the value of packet size, sending rate. Following Table 9 is the detail specification for testing simulation.

Table 9. The configuration variables of Pareto traffic.

Name	Default	Setting Value	Test Range
packetSize_	210 bit	1500 bits	1300,1400,1500,1600,1700 bits/sec
burst_time_	500 msec	0.7 sec	Not change
idle_time_	500 msec	0.3 sec	Not change
rate_	64 Kbit/sec	7500 bits/sec	7300,7400,7500,7600, 7700 bits/sec
shape_	1.5	Default	Default

3.5.1.4. Poisson Traffic

Poisson traffic generator is not directly supported by NS-2. In the NS-2 manual, it describes the way to generate Poisson traffic with Exponential traffic generator [25]. But Exponential traffic generator does not generate Poisson traffic reasonably. And we will use Poisson traffic generator made by Kostas Pentikousis [18]. This is a patch for Poisson traffic made by an individual. We set the traffic configuration like Exponential. We chose a utilization of 0.85, sending rate is five packets/second and service rate is 5.88 packets/second. Also, we change the value of two configurations like other traffic generator: packet size and sending rate. Table 10 shows the detail specification for testing the simulation.

Table 10. The configuration variables of Poisson traffic.

Name	Default	Setting Value	Test Range
rate_	1 Mbit/sec	7500 bits/sec	7300,7400,7500,7600, 7700 bits/sec
interval_	1 sec	Default	Default
packetSize_	500 Byte	1500 bits	1300,1400,1500,1600,1700 bits/sec
maxpkts_	$2^{28}\ packets$	Default	Default

3.5.2. Prediction Time

Stuckey estimates past network information and propagates that to the present network state; however, our modified NS-2 Kalman filter uses the present data to predict the future network state. The difference is shown in the Figure 16 and Figure 17.

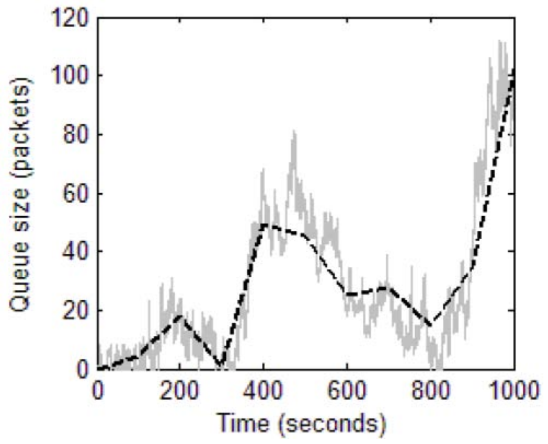


Figure 16. Estimate of queue size in packets overlaid on the actual queue size values, 100 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate [15].

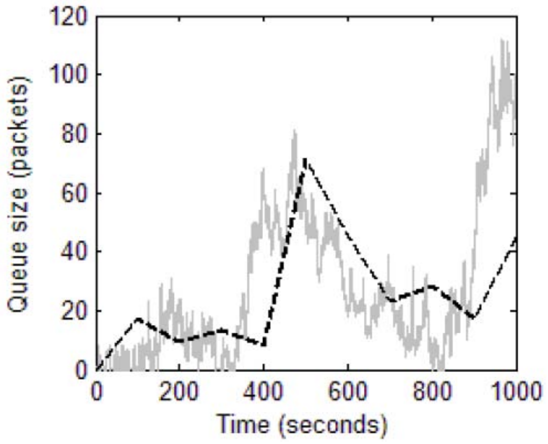


Figure 17. Prediction of queue size in packets overlaid on the actual queue size values, 100 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate [15].

The Figure 16 graph is described as Stuckey's method; he had more interest in the work of the Kalman filter than predictive ability of the Kalman filter. He discussed the predictive ability of the Kalman filter like Figure 17. However he used to validate his thesis with the graph in Figure 17. Therefore we will use simple network model to predict the queue size because we have more interest in the predictive ability of the Kalman filter.

To develop this model to validate the stochastic scheme, we predict in multiple future time intervals. This is because we need to know what prediction time is most effective when we predict with Kalman filter. In this research, we predict for 10, 20, 30, 40, 50 and 100 second samples to predict future network states. This prediction time is similar time as Stuckey validates his theory with OPNET model.

3.5.3. Simulation Scenarios

We will test with three scenarios to validate Stuckey's stochastic control scheme, some network scenarios models were designed as referenced with Stuckey's OPNET model. 1) To verify NS-2 code calls MATLAB function, we made a simple network model, and with this simple network model, we can check validation of the Kalman filter that it works with NS-2. 2) Another network model is mixed traffic model to present real network situation. Like real network. This network model has many nodes, links and traffics. 3) Last model is simple feedback controller. In this scenario, controller model estimate network state and regulates queue size.

3.5.3.1. Simple Network Estimator Model

This model is simple model to test validation of the Kalman filter with NS-2 and MATLAB. For this validation, we will try to check the cooperation between NS-2 and MATLAB as described previous section. Then, we will validate the Kalman filter model under various network situations. This model will be able to verify the Kalman filter work; in addition, we can know the performance of the Kalman filter under various network traffic types.

This scenario has one source nodes and one sink node. In Figure 18, n1 is the traffic generator node and n0 is the sink node. The source node has own network traffic model and queue model. This traffic model is described as previous section and is attached to UDP traffic to transfer. This network traffic model has different traffic packet size, interval time, traffic rate and type of queue as described previous section to verify the Kalman filter. The sink node is just provided as destination of this network. This scenario is designed from Stuckey's OPNET model of the simple queueing system [15].

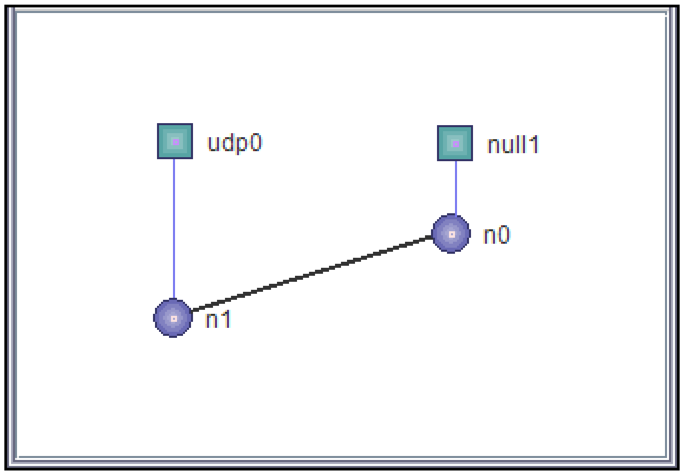


Figure 18. Integrated simple network model.

3.5.3.2. Mixed Network Estimator Model

This model is mixed network model to test validation of the Kalman filter under imaginary real network. For this validation, we will test the performance of the Kalman filter model under multiple traffic sources like real network. This model also will be able to verify the Kalman filter work; in addition, we can know where the best performance of the Kalman filter is attached different place in this model.

This scenario has multiple nodes to generate the traffic. Each node has each own traffic generator that has specific traffic characteristics. As shown in Figure 19, many nodes and links are mixed and they have their own traffic type. In this model, only one node is sink node. Each node does simple routing on the base of the distance vector. This scenario is designed from Stuckey's OPNET model of the mixed traffic network [15].

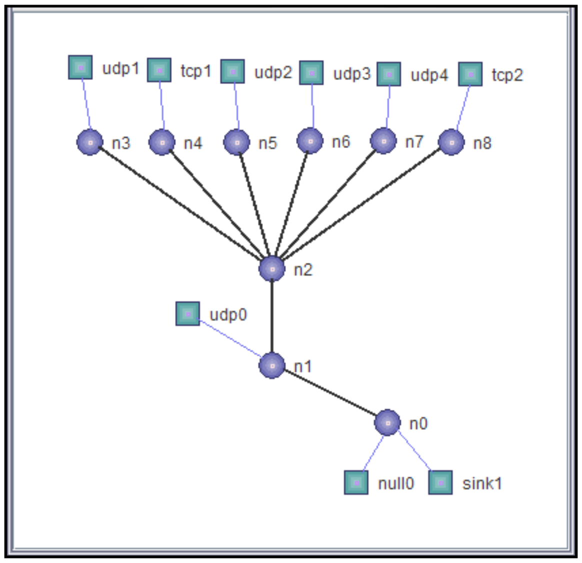


Figure 19. Integrated mixed network model.

3.5.3.3. Simple Controller Model

This model is simple controller model to test the work of queueing controller with Kalman filter. For this validation, we will test the performance of controller model under simple network. This model will be able to verify the Kalman filter work; in addition, we can know how the performance of queue is improved by the Kalman filter controller.

This scenario is similar as the simple network estimator model except using feedback controller model. As shown in Figure 20, the dashed lines are statistic lines that report the queue size and queue arrival rate to the source. With this report, the controller controls the traffic source to regulate the service. That is, this controller predicts the future network state – queue size – with Kalman filter and control packet arrival rate of the source. This scenario is designed from Stuckey's simple controller model [15].

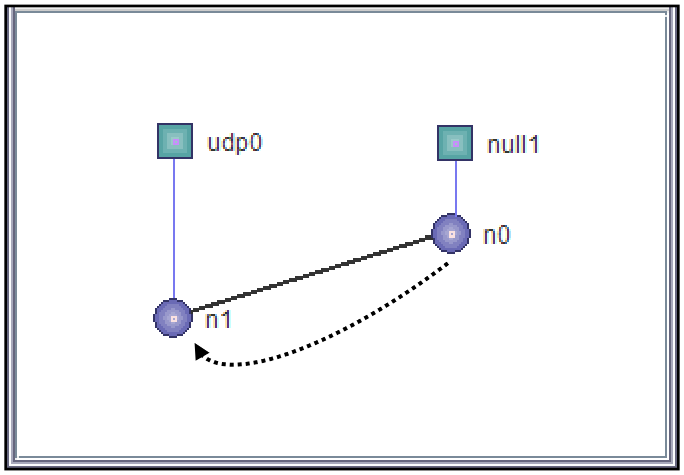


Figure 20. Integrated feedback controller model.

3.6. Summary

In this chapter, method for validation of porting into NS-2 is developed. And the various models are developed to verify the performance of Kalman filter estimator and controller under ideal real network. In these models, we chose type of traffic, packet size, traffic rate and prediction time. These network configuration help to add the validation for Stuckey's stochastic theory. The performances of these designs are explored in the following chapter.

IV. Result

4.1. Introduction

This chapter presents the result of the validation of our stochastic control scheme designed with a Kalman filter. For this validation, we designed models in chapter Ш and, we validate cooperation of NS-2 and MATLAB. We describe the details of the model implementations and results to validate stochastic scheme.

4.2. Validation

As described in Section 3.4, in this section, we validated the NS-2 with two parts. First, we validated the network model developed by NS-2. The second, we validated the cooperation between NS-2 and MATLAB.

4.2.1. Validation of NS-2 models

First of all, to validate the queue model in NS-2, we designed the same network model as Stuckey's validation model. However, Stuckey used the OPNET model and we will use the NS-2 model. We chose a utilization of 0.85 to load the queue sufficiently in order to provide a reasonable dynamic response. A sending rate of five packets/second was chosen as a reasonable sending rate, resulting in a service rate of 5.88 packets/second. This all configuration figure is same as Stuckey's model.

Figure 21 is the ideal result under above network configuration that is under $\rho = 0.85$ and $\mu = 5.88$ packets/second.

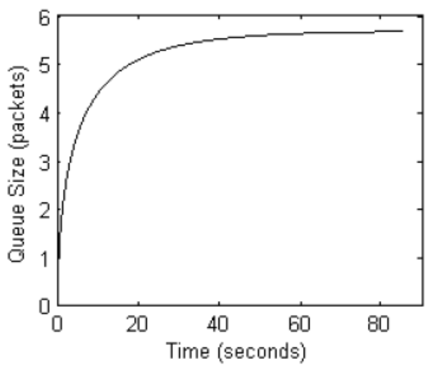


Figure 21. The theoretical expected queue size is shown for a utilization of 0.85 [15].

To validate this model, Stuckey used the network simulation tool OPNET modeler version 11.5 and we use the NS-2 tool version 2.29. Figure 22 and Figure 23 shows the design of the simple queueing model for this validation. Figure 23 is represents the queueing model in NS-2 with visualization tool NAM [14].

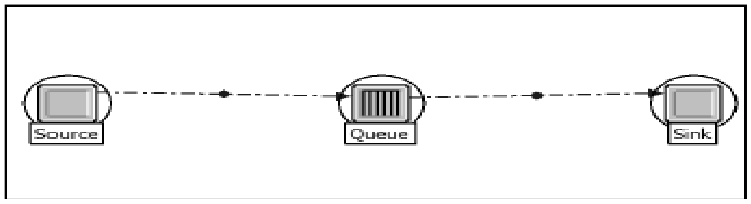


Figure 22. OPNET model to validate the queueing model [15].

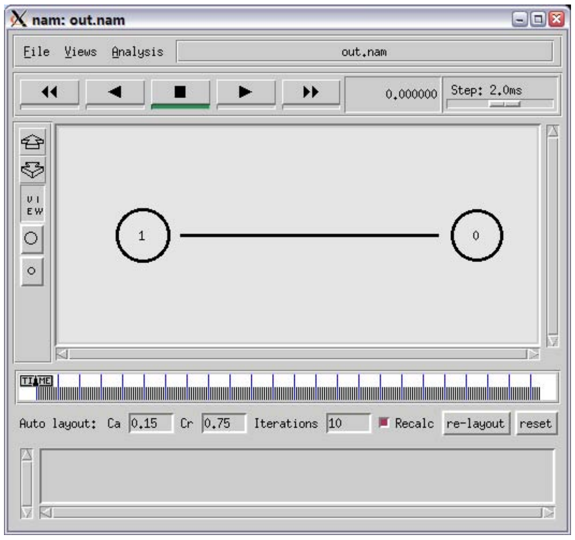

nam: out.nam
File
Views
Analysis
out.nam
0.000000
Step: 2.0ms
VIEW
1
0
TIME
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc re-layout reset

Figure 23. NS-2 model to validate the queueing model shown in NAM.

This validation model consists of a source node, a sink node and queue. A source node generates Poisson traffic with exponentially distributed packet sizes and a sink node just receives these packets. The queue services each bit of the received traffic at a constant rate, which is due to the exponentially distributed packet size, and this results in an exponential service rate. The mean packet size is 1500 bytes. The service rate of the queue was set to 70,588 bits/second in order to match the service rate of 5.88 packets/second with a utilization of 0.85 as shown in Figure 21.

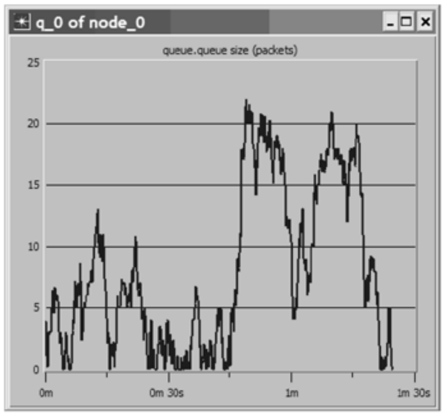


Figure 24. The number of packets contained in the queue is shown for a single simulation run of the simple queue model with a utilization of .85 and service rate of 5.88 packets/second in OPNET [15].

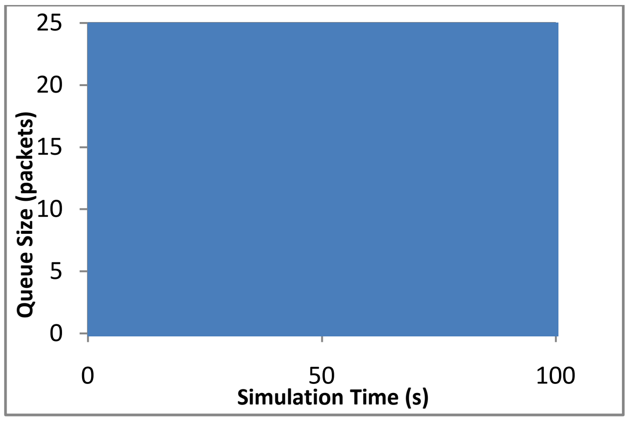


Figure 25. The number of packets contained in the queue is shown for a single simulation run of the simple queue model with a utilization of .85 and service rate of 5.88 packets/second in NS-2.

Figure 24 and Figure 25 displays the results from a single run of the validation model in each NS-2 and OPNET. The results can vary greatly for different runs using different random seed values to generate the Poisson traffic. As a result, the output graphs of OPNET and NS-2 are not matched precisely because of the effect from a different random seed, however the form of the graphs are similar. To compare the results obtained to the expected value results shown in Figure 21 properly, a Monte Carlo analysis was conducted using multiple seed values.

To compare typical queue behavior to the theoretical calculations, an eleven run Monte Carlo analysis was performed. As shown in Figure 26, the mean and one standard deviation confidence bounds of the queue size are calculated automatically by OPNET tool. The NS-2 does not have the ability to calculate with Monte Carlo analysis and so we used Excel to calculate the mean with output of NS-2 [6] [13].

Notice that the mean value of Figure 26 and Figure 27 matches very closely with the ideal results shown in Figure 21. As described above the individual results will not be predicted precisely, however the mean of many runs match the calculated ideal value closely. The results from OPNET and NS-2 validate that the expected queue size is correct under ideal conditions. Also we can trust the NS-2 queueing model by this validation.

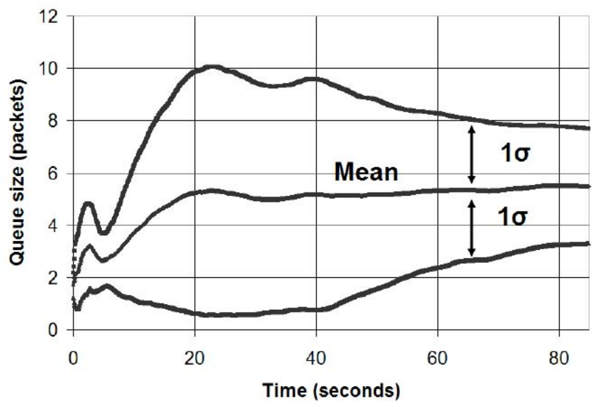


Figure 26. Eleven-run Monte Carlo results obtained from the simple queue model with a utilization of .85 and a service rate of 5.88 packets/second in OPNET [15].

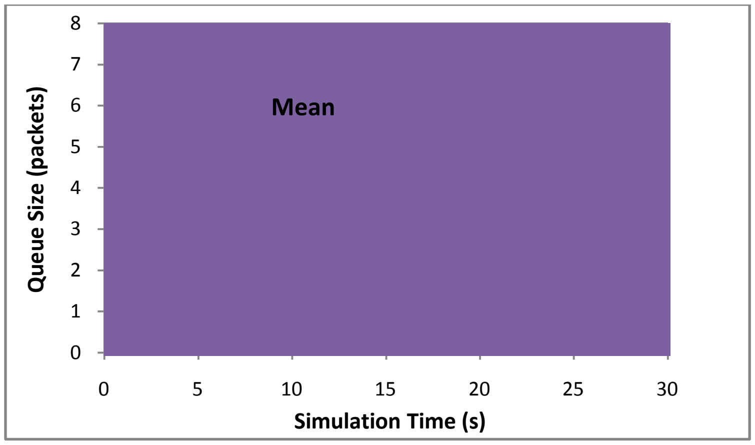
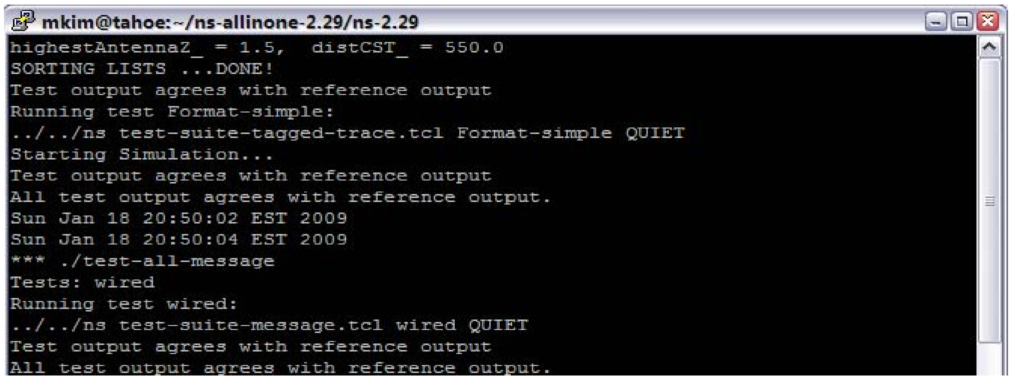


Figure 27. Eleven-run Monte Carlo results obtained from the simple queue model with a utilization of .85 and a service rate of 5.88 packets/second in NS-2 [15].

4.2.2. Validation of cooperation between NS-2 and MATLAB

Next we will validate the cooperation between NS-2 and MATLAB. As described above, the Kalman filter is written by the MATLAB, and the NS-2 code called the MATLAB engine to calculate the predicted queue size. We can check this validation with the test suite support provided by the NS-2 simulation. The NS-2 distribution contains many test suites under ~ns/tcl/test, which are used by validation programs (~ns/validate, validatewired, validate-wireless, and validate.win32) to verify that the installation of the NS-2 is correct. This is because we modified and added new modules to the NS-2, and we can run the validation programs to make sure that our changes do not affect other parts in the NS-2.



```
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
Test output agrees with reference output
Running test Format-simple:
../../ns test-suite-tagged-trace.tcl Format-simple QUIET
Starting Simulation...
Test output agrees with reference output
All test output agrees with reference output.
Sun Jan 18 20:50:02 EST 2009
Sun Jan 18 20:50:04 EST 2009
*** ./test-all-message
Tests: wired
Running test wired:
../../ns test-suite-message.tcl wired QUIET
Test output agrees with reference output
All test output agrees with reference output.
```

Figure 28 Test suite support provided by NS-2 simulation.

However, this checking does not verify the correctness of results from cooperation between NS-2 and MATLAB. Therefore we made output data that has only actual queue size as described in Section 3.4. The Kalman Filter class makes an output data file with actual queue size and predicted queue size calculated by the MATLAB engine in Section 3.3.3. We extract only the actual queue size from this output data file. Now with this set of actual queue size, we input MATLAB code to predict queue size. Comparing the result from the NS-2 simulation and the result that is calculated by MATLAB directly, we can check whether the NS-2 calls the MATLAB engine and calculates correctly or not.

Table 11. Compare the result from the NS-2 simulation and MATLAB.

NS-2 call MATLAB				MATLAB directly			
Time	Actual queue	Time	Predicted queue	Time	Actual queue	Time	Predicted queue
10	89	20	118.41	10	89	20	118.41
20	0	30	3.12	20	0	30	3.12
30	58	40	30.25	30	58	40	30.25
40	55	50	43.83	40	55	50	43.83
50	105	60	116.36	50	105	60	116.36
60	201	70	238.30	60	201	70	238.30
70	180	80	210.59	70	180	80	210.59

Table 11 shows a perfect match with the result from NS-2 and MATLAB. This validates that the network information is sent from the NS-2 to the MATLAB engine and MATLAB engine calculates to predict queue size correctly.

4.3. Result of Simple Network Estimator Model

Stuckey's primary goal was to generate an accurate estimate of queue size [15]. The measurement data was generated by the mixed traffic OPNET simulation. The current queue size in this model was sampled and the data file was manually transferred to MATLAB. In MATLAB, Stuckey added noise with a variance of R = 10 to the sampled data to simulate the measurement uncertainty as a real network. This measurement uncertainty is due to the lack of a global time reference. Because the measurement time is not known exactly, the noise of measurement is directly influenced by the accuracy of the clocks in the real network. An R = 10 was estimated by Stuckey to provide this measurement noise. In our simulation, we used the same noise variance to provide the measurement uncertainty.

We need to talk about another variance to achieve the best performance of the Kalman filter. This is the Q_d variance (matrix) that tunes the filter properly. To obtain a proper estimate, the filter must be tuned and this is done by adjusting Q_d in an iterative manner until the best performance is achieved. Initially, when the sample rate was set to 10 seconds, Stuckey found a proper tuned variance with a matrix of $Q_d = \begin{bmatrix} 20 & 0 \\ 0 & 0.01 \end{bmatrix}$. In our simulation, this sample rate is equal to prediction time. This is because with network information for sample rate, we calculated the queue size after this sample rate as prediction time. Therefore we used the same tuned variance Q_d to simulate our models.

Figure 29 shows the estimate queue by Stuckey with these variances. The estimate is overlaid on the actual queue size value. And Figure 30 is the estimated queue by our simulation with the same method and variance.

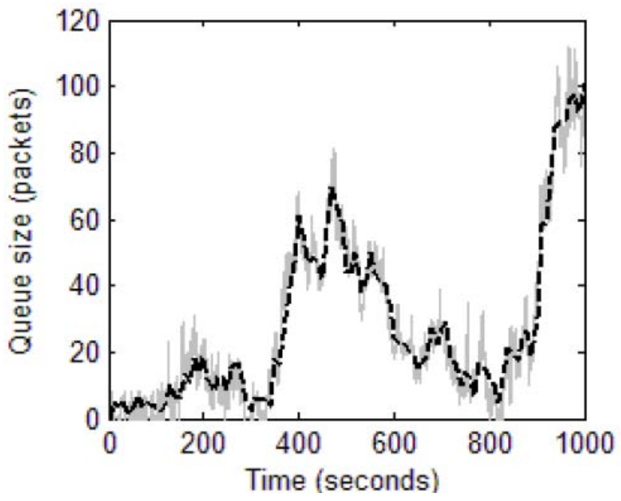


Figure 29. Estimate of queue size in packets overlaid on the actual queue size values by Stuckey's OPNET [15], 10 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate.

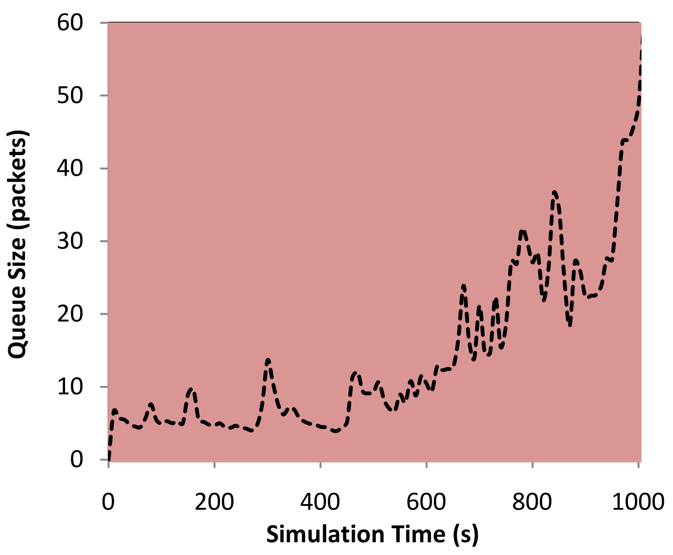


Figure 30. Estimate of queue size in packets overlaid on the actual queue size values by NS-2, 10 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate.

Of course, the shapes of these two resulting graphs are different. This is because of the burstiness of the Poisson traffic and the differences of model between the OPNET and NS-2. Therefore we need to notice how the estimate of queue size follows the actual queue size. Both of them are provided by the network estimator with Kalman filter and do a very good job following the actual queue size. With these results, we can know the accuracy of estimator of Kalman filter and also we can validate that ported Kalman filter into NS-2 with MATLAB. That is, the graph shown in Figure 30 verifies the simulation work within Kalman filter within NS-2.

Next we ran the simulation with sample rate 100 seconds to see the effect of reducing the sample rate. The iterative filter tuning procedure described earlier was conducted and Stuckey chose $Q_d = \begin{bmatrix} 30 & 0 \\ 0 & 0.01 \end{bmatrix}$ to obtained best performance. Also he used the same noise variance R = 10. The following Figure 31 and Figure 32 show the result of this simulation and our simulation with the same variances as Stuckey's

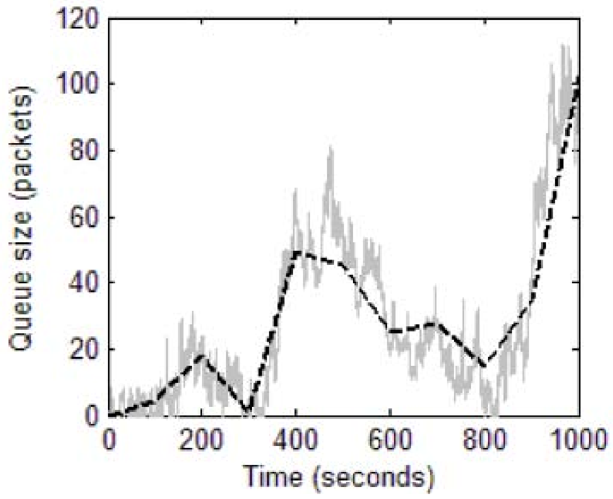


Figure 31. Estimate of queue size in packets overlaid on the actual queue size values by Stuckey's OPNET [15], 100 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate.

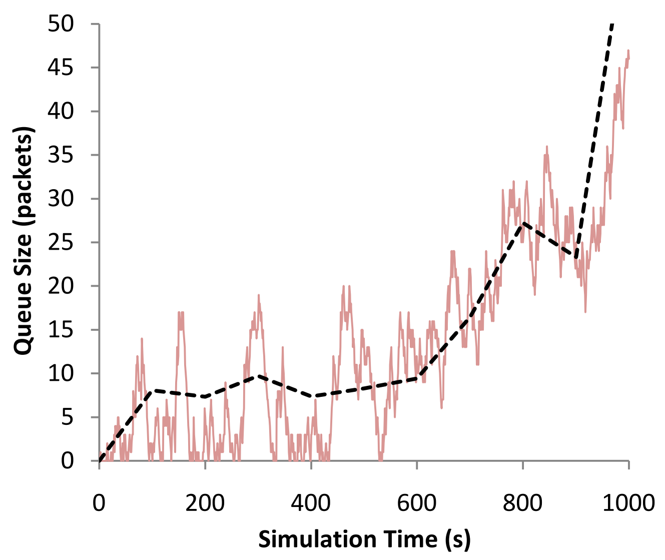


Figure 32. Estimate of queue size in packets overlaid on the actual queue size values by NS-2, 100 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate.

As described in Figure 29 and Figure 30 graph with sample rate 10 second, even at a much reduced sample rate, the Kalman filter estimates the queue size adequately. It is obvious to increase difference between the actual queue size and the predicted queue size. However with the 10 second sample rate, the estimate of queue size does follow the actual queue size. This also verifies the simulation works with Kalman filter within NS-2.

4.3.1. Type of Traffic

Until now, we discuss the work within Kalman filter in Stuckey's thesis. The accuracy and the estimate of Kalman filter but from now on we discuss the predictive ability of Kalman filter. As described in Chapter III, we tested several models by the type of network traffic. This is because the real network has very many traffic types. As a result, we tested four types of traffic: Exponential, CBR, Poisson and Pareto. Each type of traffic has a specific character that is represented character of real network. The purpose of this simulation is to know the difference of prediction with Kalman filter by the type of the network traffic.

To run this simulation, we used simple network estimator model described in Section 3.5.3.1 and also used the same variance described earlier (R, Q_d). Then each configuration of each traffic type described in Section 3.5.1 was chosen. The following graphs are the results showing the difference of prediction by the different type of the network traffic.

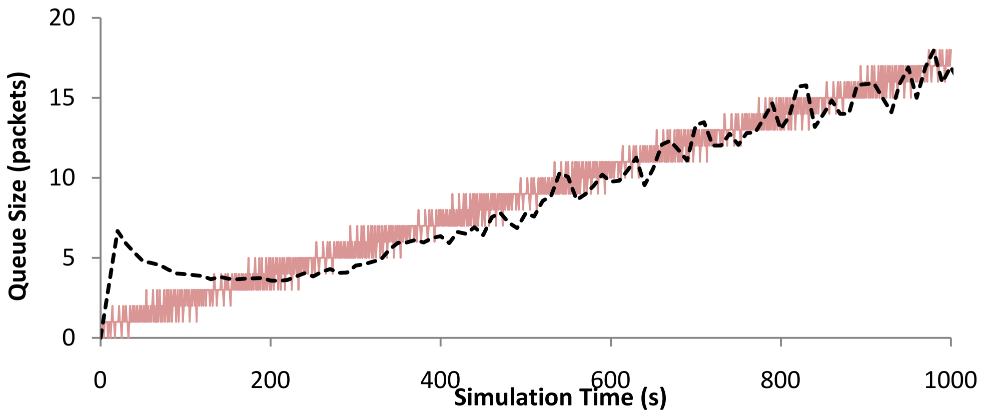


Figure 33. Estimate of queue size by the type of CBR traffic. 10 second sample period, rate: 7500 bits/sec, packet size: 1500 bits, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate.

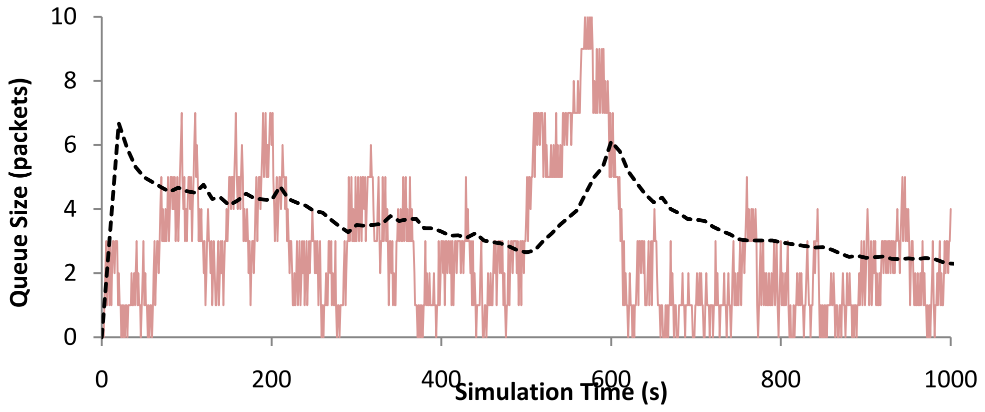


Figure 34. Estimate of queue size by the type of Exponential traffic. 10 second sample period, rate: 10000 bits/sec, packet size: 1500 bits, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate.

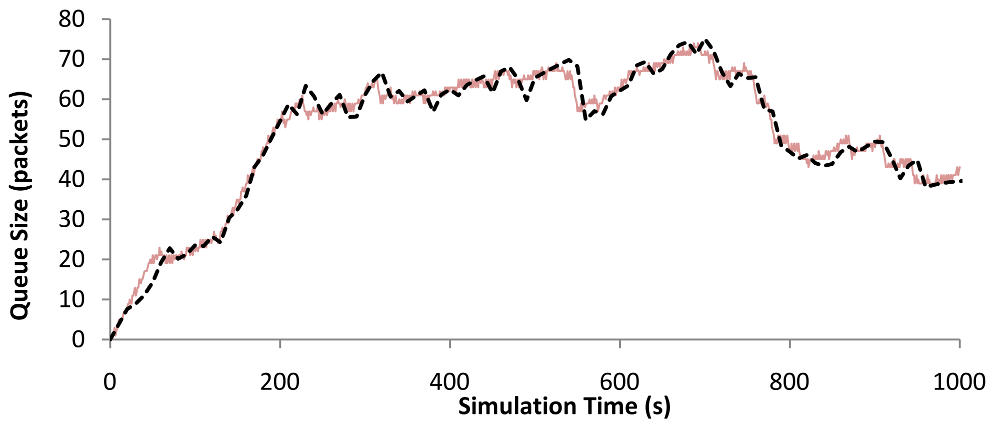


Figure 35. Estimate of queue size by the type of Pareto traffic. 10 second sample period, rate: 10000 bits/sec, packet size: 1500 bits, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate.

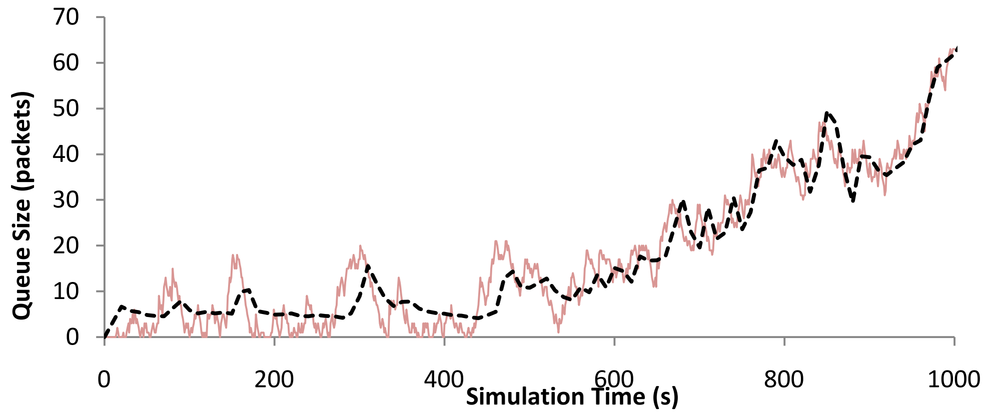


Figure 36. Estimate of queue size by the type of Poisson traffic. 10 second sample period, rate: 7500 bits/sec, packet size: 1500 bits, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate.

These graphs are shown estimate of queue size by the type of traffic. The dashed line is the prediction queue size and the grey line is actual queue size. All graphs of the estimate to predict queue size did well in following the actual queue size as describe earlier. One problem is the initial difficulty in predicting from zero start. In NS-2 with MATLAB, if the actual queue size is less than predicted queue size, the Kalman filter calculated the predicted queue size with zero actual queue size is 6.68198 packets. On the other side, if the actual queue size is greater than predicted queue size, the Kalman filter calculated normally like OPNET. As a result, except the initial increase in predicted queue size, the Kalman filter adequately follows the form of the actual size. Table 12 shows the results by statistical method.

Table 12. The statistic result by MSE, MAPE.

Traffic type	From 0 second		From 100 second	
	MSE	MAPE	MSE	MAPE
CBR (fig 33)	2.49	19.34 %	1.68	15.71 %
Exponential (fig 34)	4.76	41.64 %	3.99	40.07 %
Pareto (fig 35)	8.87	5.5 %	8.27	4.1 %
Poisson (fig 36)	25.07	44.25 %	25.26	41.03 %

This Table 12 shows more distinctly the performance of the Kalman filter. In this result, the best performance of the different traffic types is CBR and Pareto traffic. CBR has the smallest error of MSE and has small MAPE with the second time. In MAPE, the Pareto showed the best performance. As shown in Figure 35, the predicted queue size was very good at following the actual queue size than others. MAPE is calculated by dividing by the standard value (predicted queue size), and the Pareto could show the best result in MAPE. But, in MSE the Pareto value had more queue size than CBR. Therefore the result of Pareto had more absolute error in MSE. However this is not enough to verify the performance of the Kalman filter. To have more validation, we tested this simulation with a Monte Carlo method. The following table is the statistic result from ten run simulations with exponential distribution packet size. The MSE and MAPE in this table are the average MSE and MAPE for the ten run simulations. And following graphs show the average actual and predicted queue size for ten run simulations.

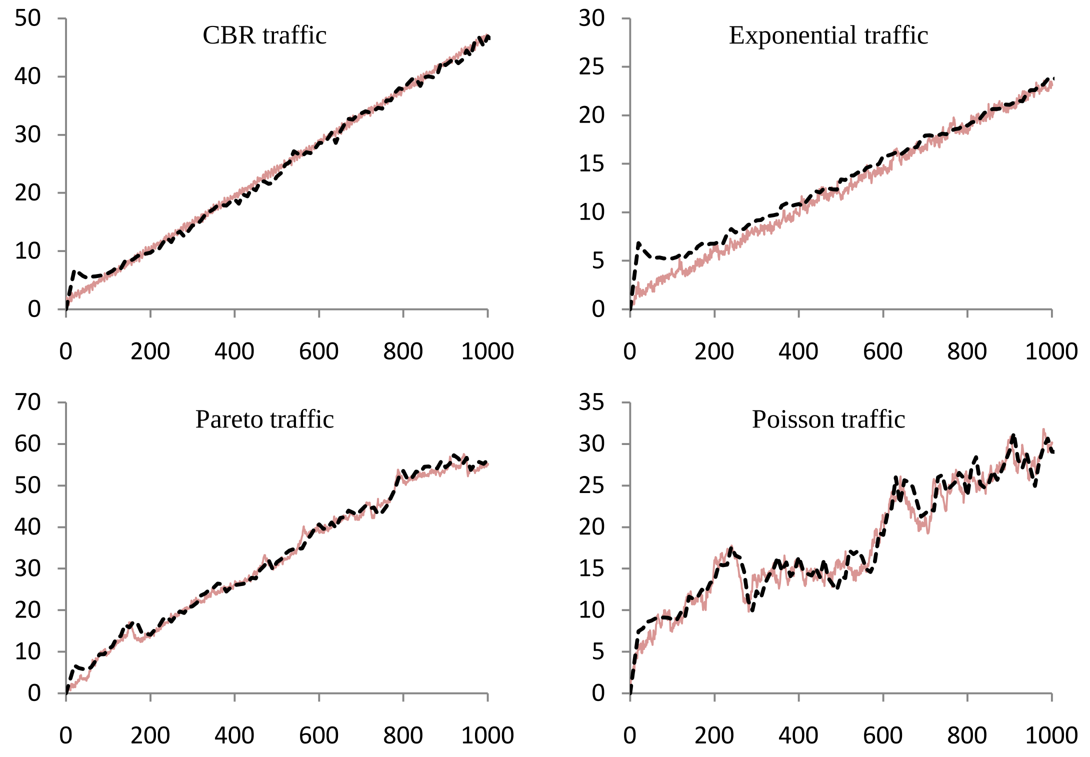


Figure 37. Ten run Monte Carlo estimate of queue size by the traffic type. 10second sample period, rate and packet size is same as above graphs, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate, X-axis: queue size (packets), Y-axis: simulation time.

Table 13. Ten run Monte Carlo statistic result by MSE, MAPE.

Traffic type	From 0 second		From 100 second	
	MSE	MAPE	MSE	MAPE
CBR	3.53	6.27 %	3.06	4.52 %
Exponential	4.01	10.31 %	3.23	7.14 %
Pareto	15.56	5.81 %	15.25	4.03 %
Poisson	23.17	10.3 %	23.44	10.03 %

As discuss earlier, the best performance of the traffic type was the CBR and Pareto traffic. The CBR traffic has a simple traffic generator and is easy to predict future size. Also, the CBR traffic does not have any burstiness, and the CBR traffic maintains queue size between zero and one, or increases infinitely as the same rate. Actually, the CBR traffic is able to be predicted without stochastic method. For these reasons, the Kalman filter predicted the CBR traffic very well. As a result the CBR traffic showed the good performance in MSE and MAPE.

In the case of Pareto traffic, the graph of one run simulation with Pareto traffic was simple than other graph (Exponential and Poisson). So, it was easy to predict the queue size by Kalman filter. Actually, with the naked eye the graph of Pareto traffic was best following the actual queue size. But, the Pareto traffic had much more traffic than other traffic type. In the MSE with absolute error, the Pareto traffic showed lower result than other traffics. However, it was true that the Pareto traffic had best performance in MAPE as shown Table 13.

Next performance of traffic type is Exponential traffic and lastly Poisson traffic. The Poisson had also large actual queue size and frequency, and as a result, it showed the lowest performance. The Exponential traffic was similar as the Poisson traffic but the Exponential traffic had less traffic than Poisson traffic and it had the less MSE.

Also we notice the initial errors that were described earlier. The initial error occurs in the small actual queue size. So Table 12 and Table 13 distinguish performance of the Kalman filter from initial error. The statistic result includes the initial error is calculated from zero but the other result without initial error is calculated form 100 second. Comparing these results, we can know that the performance of Kalman filter is increased without initial error. In case of Exponential traffic, its performance is increased 3.1 % and the Poisson traffic improved by 0.2 %. The Poisson traffic did not have any improvement in Table 13 because it had more actual queue size regardless of the initial error.

Until now, we have discussed the performance of the different traffic. With this result, we can know the performance of the Kalman filter by the stochastic method. The Kalman filter has more efficiency when the actual queue size is small and has simple frequency. In this simulation, we found the best performance as the CBR and Pareto traffic. However, as with the network configuration, the other traffic type has better performance. Accordingly, the best performance is less important in this simulation. It is more important that we can predict the actual queue size under 4 ~10 % relative error.

4.3.2. Prediction Time

As describe in Chapter III, we tested the model designed with Kalman filter by changing prediction time. This prediction time is the sample rate to calculate future queue size with current network state information. So, we tested this model with Poisson traffic and prediction time is 10, 20, 30, 40, 50, and 100 sec.

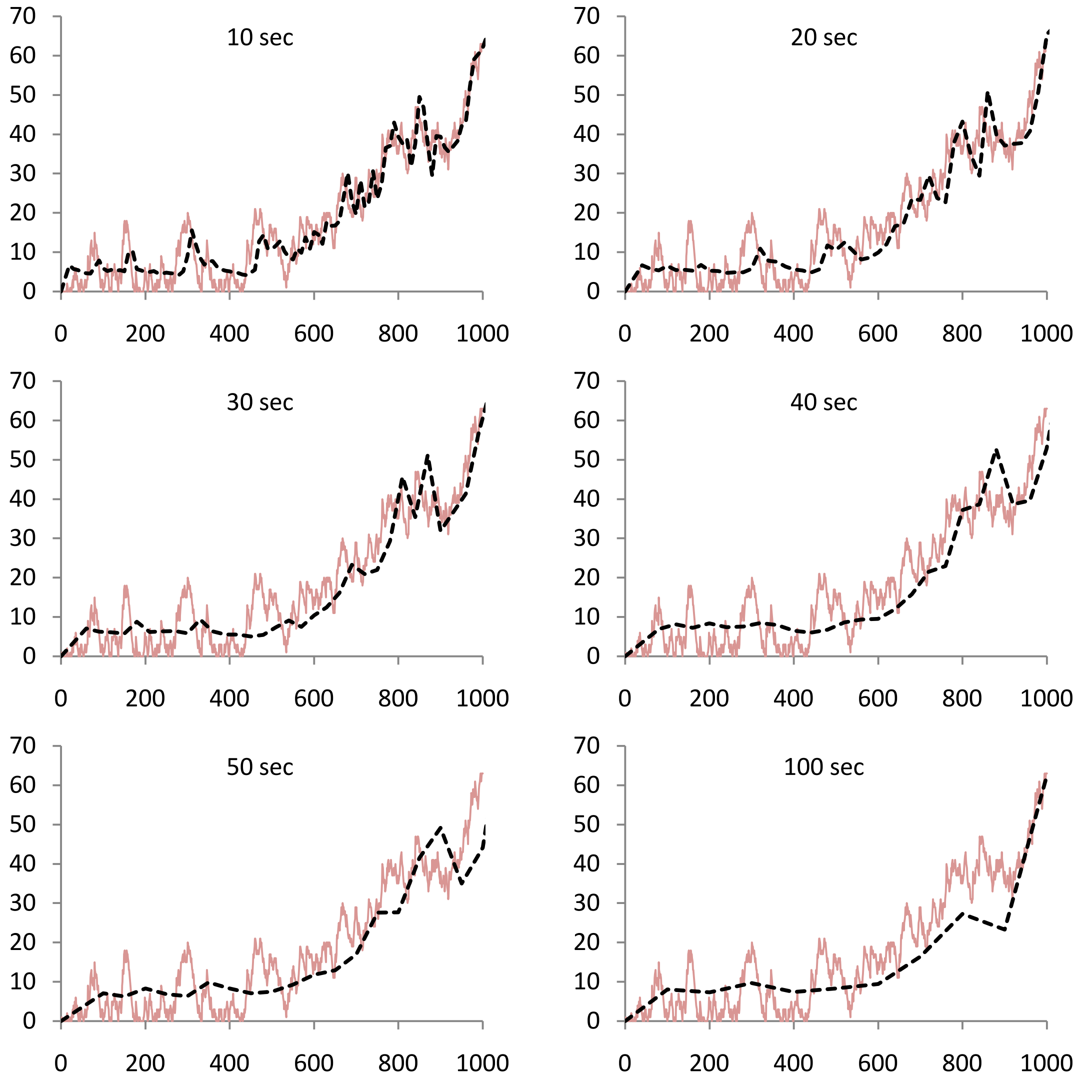


Figure 38. Estimate of queue size by the type of Poisson traffic. 10, 20, 30, 40, 50 and 100 second sample period, rate: 7500 bits/sec, packet size: 1500 bits, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate, X-axis: queue size (packets), Y-axis: simulation time.

These graphs show the result of changing prediction time. The red line is prediction queue size and the grey line is actual queue size. The actual queue size is estimated by designed queue and the predicted queue size is calculated by Kalman filter as described above. As shown in this Figure 38, the smaller the time that we predicted, the more accurate is the output that we can get. Following Table 14 is statistic result by MSE and MAPE by ten run Monte Carlo analysis.

Table 14. The ten run Monte Carlo statistic result by changing prediction time.

Traffic Type	Prediction Time	MSE	MAPE
Poisson	10	23.17	10.31 %
	20	66.43	11.77 %
	30	150.38	14.15 %
	40	216.11	17.61 %
	50	289.18	22.44 %
	100	579.65	26.25 %

As shown in Table 14, the more prediction time we calculate queue size at, the more error is generated between actual queue size and predicted queue size. It is obvious that if we would predict a greater future queue size, we would have fewer samples and we could not calculate queue size accurately with Kalman filter. In MAPE, we can know that these differences between actual queue size and predicted queue size have 10 % ~ 26 % scale error. And in this Poisson traffic, we do not care about the initial error by NS-2. As mentioned previously, in this Poisson model the initial error does not have much effect.

In this section, we show that the performance of Kalman filter has better performance with small prediction time, and the estimate of queue size provided by the

Kalman filter with 10sec prediction time does a very good job of following the actual queue size. However the estimates of queue size over 50sec prediction time have greater difference as increasing prediction time. This is because it is impossible to predict a sudden change in queue size without measurements to indicate the changing actual queue size for prediction time. The following result is another validation for this result. Although the traffic type is changed, the result is same. The following graphs are the result by the type of Pareto traffic.

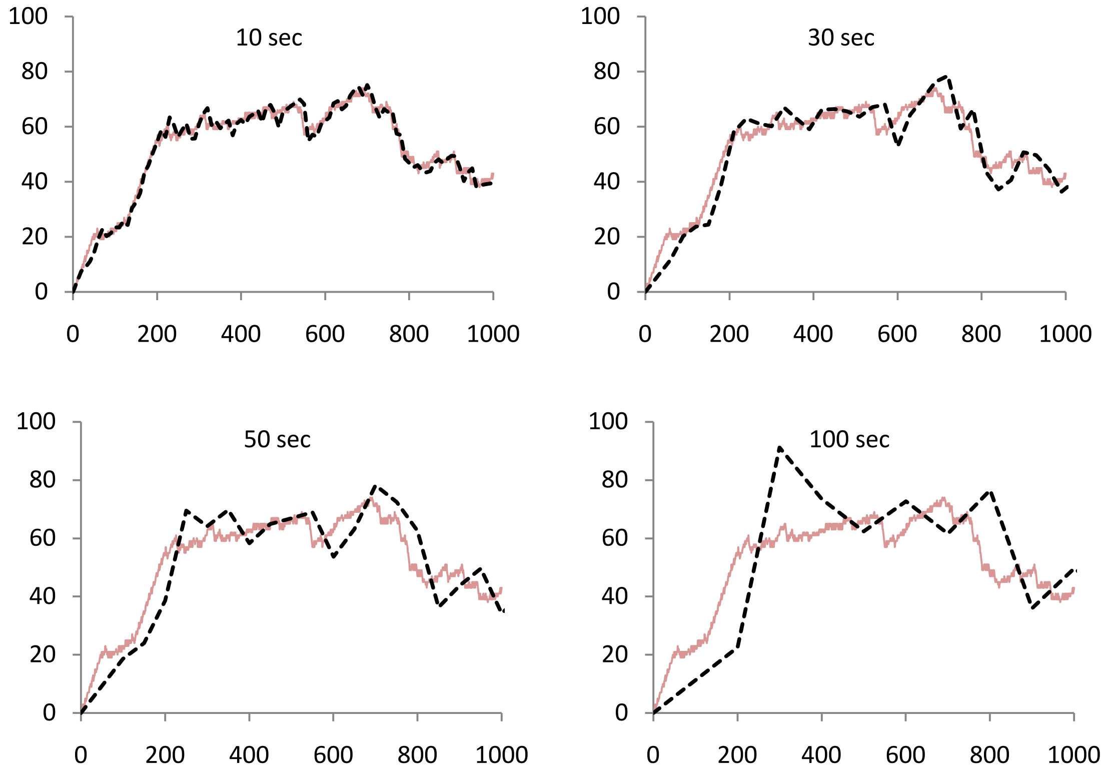


Figure 39. Estimate of queue size by the type of Pareto traffic. 10, 30, 50 and 100 second sample period, rate: 10000 bits/sec, packet size: 1500 bits, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate, X-axis: queue size (packets), Y-axis: simulation time.

The result from these graphs is same as the result of the Poisson traffic. The only difference is that graphs of the Poisson traffic have more shape frequency than the

Pareto's graphs. As a result, the prediction of the Pareto traffic is more efficient than the Poisson traffic. Also these Pareto's graphs use the prediction time to distinguish the difference as the prediction time is increased.

Table 15. The statistic result by changing prediction time.

Traffic Type	Prediction Time	MSE	MAPE
Pareto	10	8.87	5.84 %
	30	39.00	12.69 %
	50	95.23	19.94 %
	100	150.98	14.57 %

As shown in Table 15 and Figure 39, it is true that as the prediction time is increased, the performance of Kalman filter is decreased and the error is increased between actual queue size and predicted queue size. As a result, we can know the relationship between the performance and prediction time (or sample rate).

4.3.3. Packet Size

This section discusses the effect of queue size prediction by changing the packet size. The following graphs are the results made by changing the packet size. These graphs have the same configurations except for the packet size. The traffic type is Poisson. The packet size was changed by 100 bits. We wanted to know that in the high and low load network that has a big packet size, does the Kalman filter work well. This is because in Stuckey's stochastic algorithm, he tried to improve the network by controlling the queue size. Here, the packet size has big influence with queue size.

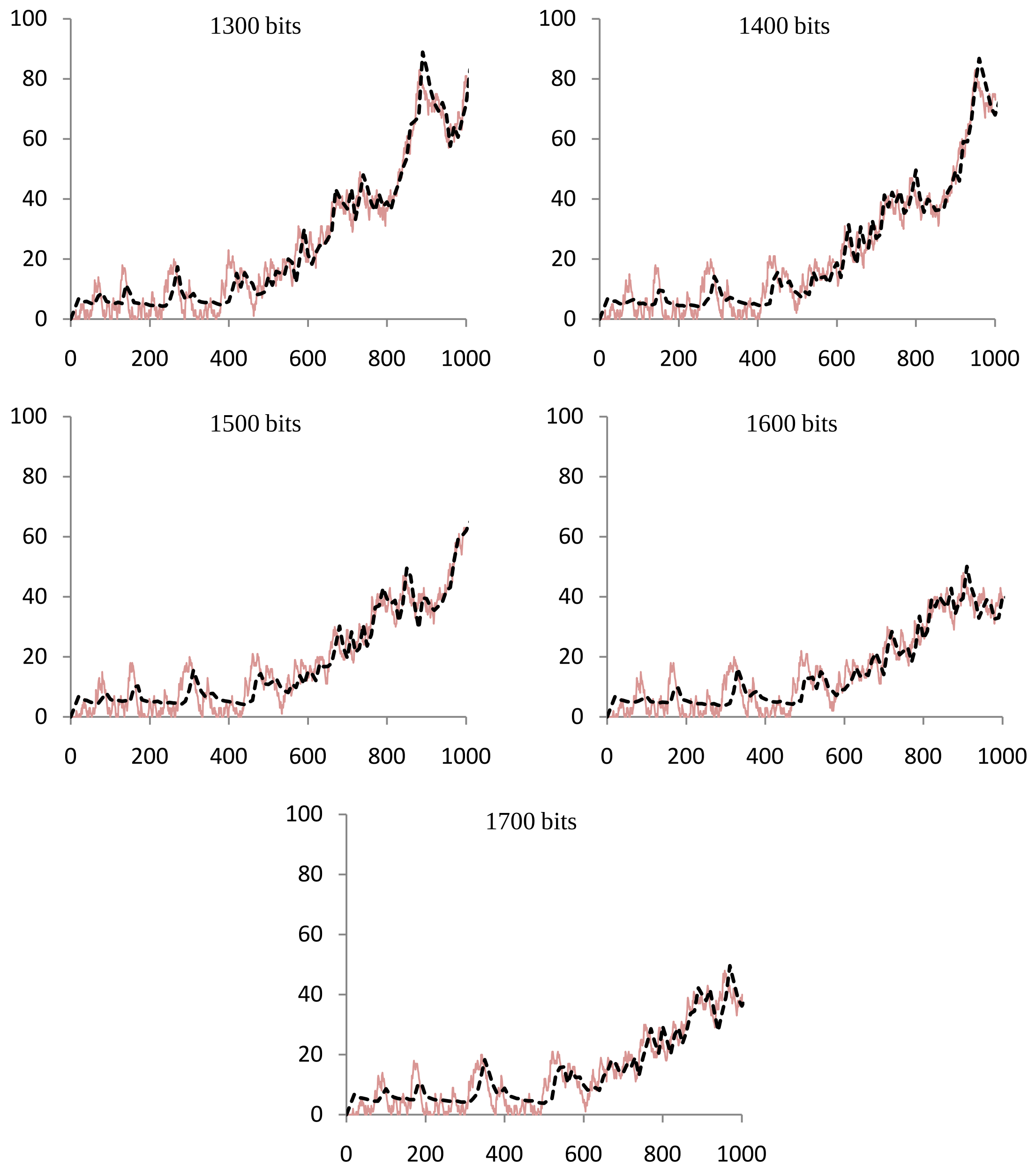


Figure 40. Estimate of queue size by the type of Poisson traffic. 10 second sample period, rate: 10000 bits/sec, packet size: 1300 bits, 1400 bits, 1500 bits, 1600 bits, 1700 bits, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate, X-axis: queue size (packets), Y-axis: simulation time.

These result graphs show the effect of the performance on the Kalman filter by changing the packet size. We notice that the graph is moved right by changing the packet size. This is because it may need more time to process the increased packet size. The following graph shows this result.

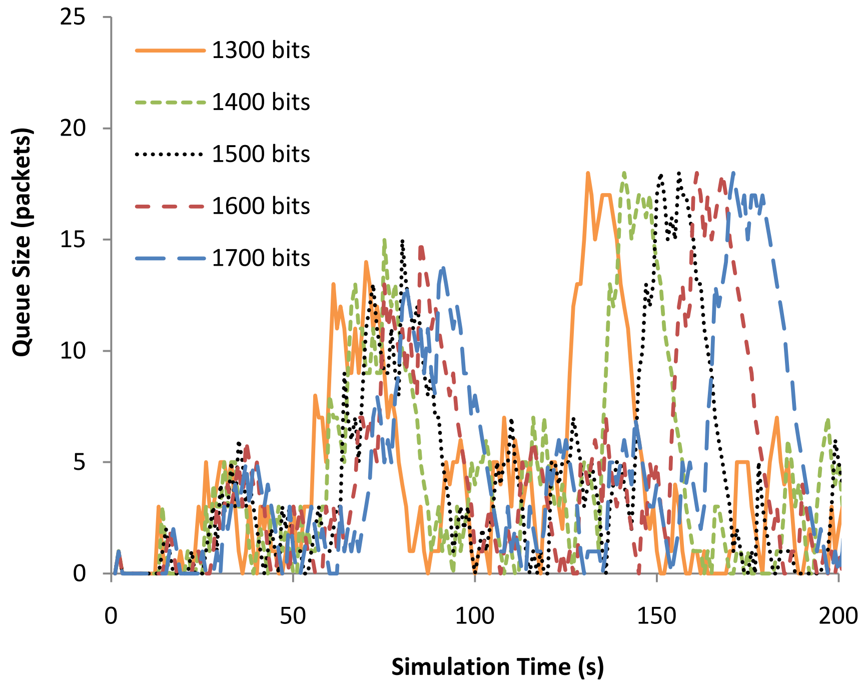


Figure 41. Actual queue size by the type of Poisson traffic. 10 second sample period, rate: 10000 bits/sec, packet size: 1300 bits, 1400 bits, 1500 bits, 1600 bits, 1700 bits.

Table 16. The statistic result by changing packet size.

Packet size	From 0 second		From 100 second	
	MSE	MAPE	MSE	MAPE
1300 bits	30.06	41.23 %	30.93	38.72 %
1400 bits	25.73	40.77 %	26.57	38.02 %
1500 bits	25.07	44.25 %	25.26	41.03 %
1600 bits	22.23	45.23 %	22.99	43.83 %
1700 bits	22.30	47.73 %	22.10	44.34 %

As shown in Figure 41, all graphs of actual queue size are almost similar as others, however it looks as if the graph moves right. This effect is shown in Table 16, and we can know that as the packet size increases, the performance of Kalman filter shows arising better tendency. Of course MAPE – relative error – does not show this tendency. Instead, the MAPE show the increasing. However it is true that absolute error is decreased as shown in MSE. This is because as the graph move right, the absolute error became decreased. However, this does influence the relative error on the contrary. This is because as the graph move right, existing graph expands, and the same relative error is expanded.

As a result, the packet size does not impact the prediction of queue size. As described earlier, the packet size influenced how many packets are stored in the queue but does not influence the estimate of queue size following the actual queue size.

4.3.4. Traffic Rate

The following topic is the sending rate of traffic. This section discusses the result of the predicted queue size by changing traffic rate. In NS-2, default measure of traffic rate is bit/sec, which is the sending bit per second time. The tested traffic type is Poisson traffic. The only traffic rate was changing by 100 bit/sec. We wanted to know that in the high traffic rate, whether the Kalman filter worked well or not. As described above section, like packet size, the traffic rate also has big influence on the network.

The result of traffic rate is similar as the result of packet size. The following Figure 42 and Table 17 is verified this result. One different thing is the contrary tendency. Here is suspected the falling tendency of performance in MSE as the traffic rate is increased and it is contrary in MAPE like result for packet size. This is shown in MSE; however like previous section, this hypothesis is not true. Actually the traffic rate also does not impact the prediction of queue size. As the graphs move right by changing the packet size, the graphs move up by changing the traffic rate. This is because the queue size is increased as the more packets are arrived for a same time. This influences the absolute error with MSE but not the relative error with MAPE on the contrary as shown in Table 17. Like packet size, traffic rate can influence the actual queue size but cannot affect the Kalman filter performance directly.

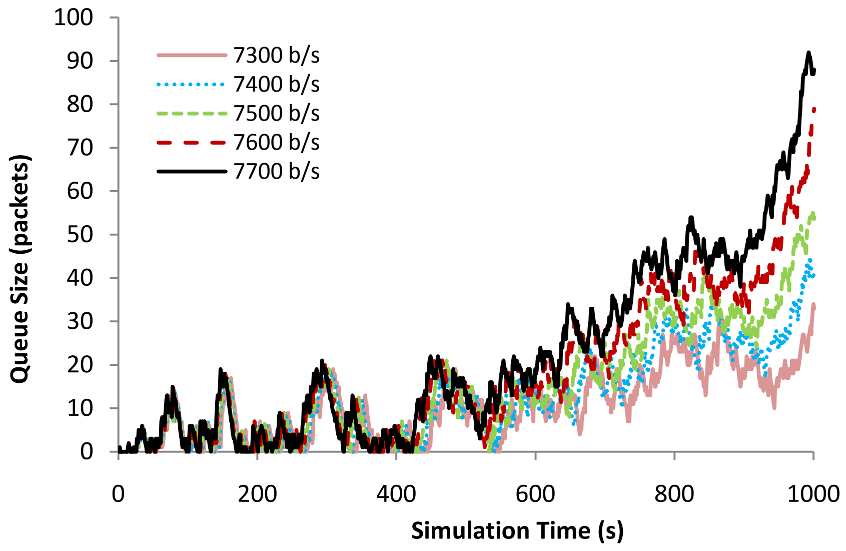


Figure 42. Estimate of queue size by the type of Poisson traffic. 10 second sample period, rate: 9800, 9900, 10000, 11000, 12000 bits/sec, packet size: 1500 bits, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate.

Table 17. The statistic result by changing packet size.

Traffic rate (sending)	From 0 second		From 100 second	
	MSE	MAPE	MSE	MAPE
9800 bits / sec	20.47	49.29 %	21.57	49.43%
9900 bits / sec	21.03	46.09 %	22.14	45.65 %
10000 bits / sec	23.90	46.48 %	23.98	43.1 %
11000 bits / sec	29.07	44.47 %	29.86	41.9 %
12000 bits / sec	25.37	41.73 %	25.57	38.16 %

4.4. Result of Mixed Network Estimator Model

At the first time, Stuckey's OPNET model of the mixed traffic network is designed from the Jackson theorem. The Jackson theorem states that, when a sufficient number of non-Poisson traffic streams are combined, they tend to become Poisson [15] [5]. So, Stuckey's OPNET model was generated with multiple non-Poisson traffic. We made a NS-2 model to this OPNET model, shown in Figure 43. As described in Section 3.5.3.2, the all nodes were used to generate traffic without one sink node. Unlike the previous scenarios, all traffic has two different types: TCP and UDP type. Therefore, at any moment in time, the traffic distributions are dependent on many variables including the performance of the transport and routing protocols in the network. Table 18 shows the detailed traffic type used by each client. Each client is validated by Section 4.2.1 when the client is designed for simple network. A reasonable cross-section of various traffic types was chosen to represent a diverse mix. Client 2, which is located in the lower right of the network in Figure 43, acted as a server to the other clients.

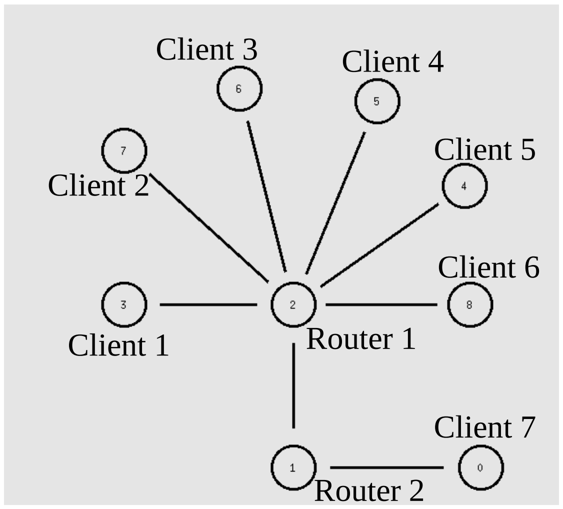


Figure 43. NS-2 model of the mixed traffic network.

Table 18. Traffic type generated by each client.

Client	Traffic Type
1	CBR(constant bit rate) traffic
2	TCP with FTP
3	Exponential traffic
4	Light Pareto traffic
5	Heavy Pareto traffic
6	TCP with Telnet
7	Simple Sink (Server to other clients)

To validate the result of the mixing model, router 1 is attached to the Kalman filter queue. In this network, the average total number of packet received by router 1 is 7386 for ten run. The average number of packets received is 7.38 packets/second. The service rate of router 1 was set to correspond to a utilization of 0.85 as described above. The following graph is obtained by the NS-2 model.

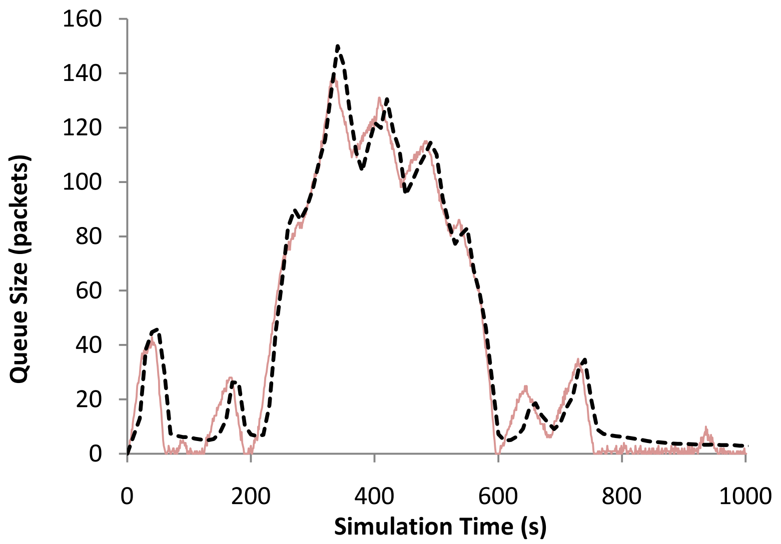


Figure 44. Estimate of queue size of router 1 by the mixed traffic model. 10 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate.

We notice that the estimate of queue size provided by the network estimator does follow the actual queue size like the simple model. Of course, while not perfect, the Kalman filter performed well in predicting the actual queue size. This result is similar with the simple model except the mixed traffic model has more of traffic. To validate the performance of Kalman filter under mixed traffic, a ten run Monte Carlo analysis was performed. And as described in the Jackson theorem, this combined traffic tends to become Poisson traffic. So we performed the Poisson traffic model to compare mixed traffic network. This Poisson traffic has the same amount of traffic as the mixed traffic network model. Table 19 shows the statistical result for this.

Table 19. The statistic result by ten run mixed network and Poisson model.

Model	From 0 second		From 100 second	
	MSE	MAPE	MSE	MAPE
Mixed network	77.36	47.51 %	70.63	47.74 %
Poisson	91.21	79.72 %	95.31	76.85 %

As shown in Table 19, the MSE and MAPE are more than the result of simple network model. This is because the amount of traffic is increased significantly. The Poisson traffic model of Table 19 is modified to generate a same amount of traffic with mixed network model. Although the total error of models is increased, this is natural when we think about the traffic amount increased in the network. It is also natural that Poisson traffic model has the more error than mixed network. This is because the Poisson traffic model has very high traffic rate and packet size to match the traffic amount at mixed network model. As a result, we can know the performance of the Kalman filter under mixed traffic network and we can also compare the performance of the Kalman filter with Poisson traffic under the same traffic condition. The performance is not as good as the simple model but we can predict the actual queue size with 47% relative error.

4.4.1. The Placement of Kalman Filter

In this section, we discuss about the placement of Kalman filter. This is the reason why the performance is changed as the placement of Kalman filter under a complicated network. To test this performance, we use the same model as with the mixed network model. This is more similar to a real network. The end of the real network consists of the router and the client is attached at the router. In this model, we tested three placements of Kalman filter: the source part, the combined part and the sink part. The source part is like a simple network estimator model. The Kalman filter queue is attached at the source node. The combined part is easy to think of as router1 in Figure 43, where all the traffic from the source is combined in this node. The Kalman filter queue is attached at the output of this node. The sink part is also easy to think of as router2 in Figure 43, this is last node to arrive at the sink node. The Kalman filter queue is attached at the outgoing of the last router before it arrives at the sink.

The following Figure 45, Figure 46, and Figure 47 are the graphs obtained by the mixed network model. Figure 46 is the graph of the combined part and Figure 47 is the graph of the sink part. The Figure 45 is source part of the graph for Poisson traffic. The other graphs of the source part can be referenced by the result graph of the type of traffic in the simple network estimator model (Section 3.5.1). As seen in these graphs, the queue size of the combined part is more than others. Basically, all predicted queue size is good at following the actual queue size. All the configuration of this section is the same as the previous section.

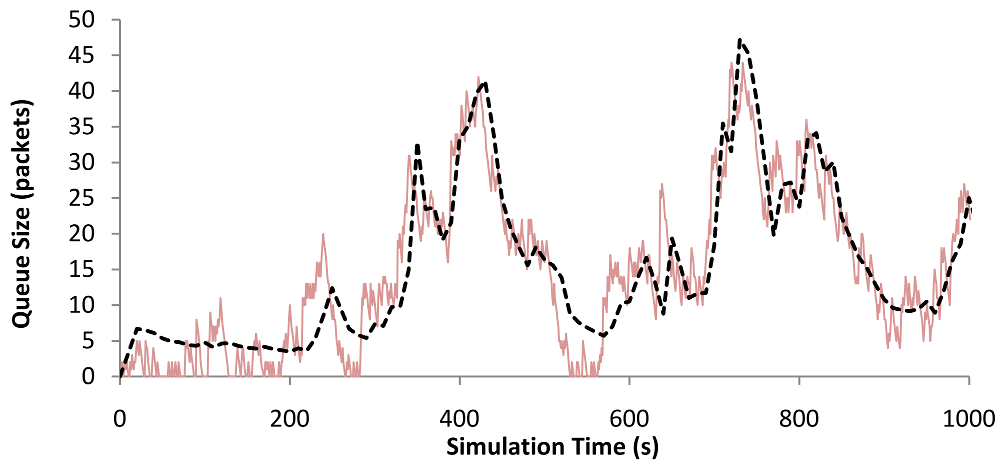


Figure 45. Estimate of queue size by the source part for Poisson traffic. 10 second sample period, grey line: actual queue size, overlaid black dashed.

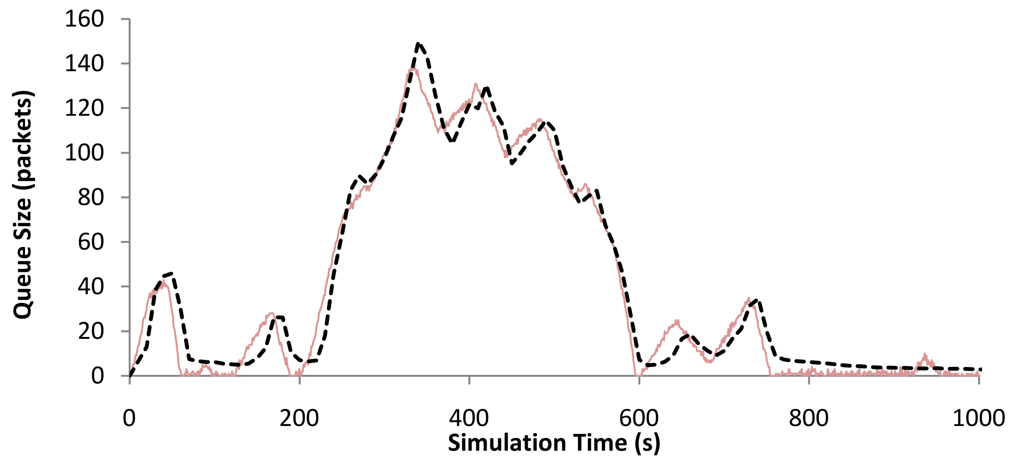


Figure 46. Estimate of queue size by the combined part. 10 second sample period, grey line: actual queue size, overlaid black dashed.

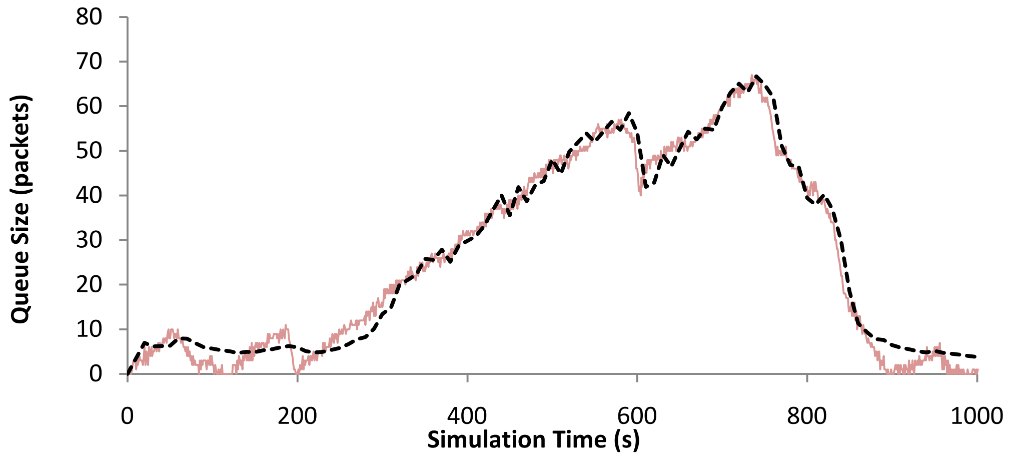


Figure 47. Estimate of queue size by the sink pat. 10 second sample period, grey line: actual queue size, overlaid black dashed.

To validate the performance for the placement of Kalman filter, a ten run Monte Carlo analysis was performed. The following Table 20 is the statistic results for this Monte Carlo analysis. In table, the source part is average MSE and MAPE for all source traffic.

Table 20. The statistic result by ten run for placement of Kalman filter.

Placement	From 0 second		From 100 second	
	MSE	MAPE	MSE	MAPE
Source	11.57	8.17 %	11.25	6.43 %
Combined	77.36	47.51 %	70.63	47.73 %
Sink	11.60	17.89 %	12.42	17.24 %

As shown in Table 20, the best performance of placement for Kalman filter is the source part. The worst performance is combined part. In the case of the source part, it has a small error in MSE, but some traffic source generated frequency traffic because of burstiness and characteristic of traffic type. Therefore the graph of the source part is more complicated than the sink part. As a result, the source part has more MAPE than the sink part. The combined part is found to have the worst performance; this is because all traffic is combined in this part and the total traffic is increased rapidly. Accordingly the MAPE and MSE are increased by the combined traffic. Moreover, this combined traffic tends to become Poisson traffic and is different from the simple traffic like CBR. It is one reason why the error is increased.

As analyzed above, the different placement of the Kalman filter has different efficiency for performance of Kalman filter. Consequently, we need to choose the placement of the Kalman filter for the best efficiency. In general, it is good to choose the source or sink part that has a smaller error than other. Although we chose the combined part to place the Kalman filter, we can expect under approximately 47 % relative error.

4.5. Result of Simple Controller Model

A controller model was designed based on the nonlinear stochastic controller theory by Stuckey. In this model, this controller was implemented in MATLAB and the simple OPNET model used to test the controller described in Section 2.7.4. Our model is designed in Section 3.5.3.3. As described in this section, the queue model was modified to control the arrival rate to the source. The source model was modified so that the sending rate was changed by the queue. Like the simple network estimator, this controller works between NS-2 and MATLAB through the co-simulation interface. In the simple network estimator, the queue calculated the queue size and in this case, the queue calculated the sending rate –traffic rate – to regulate the queue.

In the controller model, it has the tuning parameters like Q_d of the network estimator model. These parameters are $X_{11},\ X_{22},\ X_{33}$, and U. Multiple simulations were conducted to understand the effect on the performance of the controller by Stuckey. As a result, the weights X_{11} and X_{22} correspond to the importance of maintaining the states at the nominal values; the weight X_{33} corresponds to the importance of minimizing the regulation error, and the weight U corresponds to the importance of minimizing control action [15]. These parameters have the same effect on the performance of the controller in the NS-2 model, but the X_{11} and X_{22} had more error than in the OPNET model. In the OPNET model, the queue size is maintained at the same value with X_{22}. In the NS-2 model, the queue size is changed by changing the X_{22}.

Another important setpoint is y_d. This calculates the constant like $\bar{G}_c^*$ and E at the beginning of the simulation. In Stuckey's OPNET model, the setpoint was set purposely low, because small queue sizes are more difficult to maintain accurately. In this simulation, we chose the same set point 3.125 as in the Stuckey model. The parameter U was important with X_{33}. In the OPNET model, the U to X_{33} ratio of 1000 was found to provide adequate results. In the NS-2 model, the same ratio shown to have the best result of performance. The simulations were performed for 1000 seconds of simulated time with a queue service rate of 7500 bits/second.

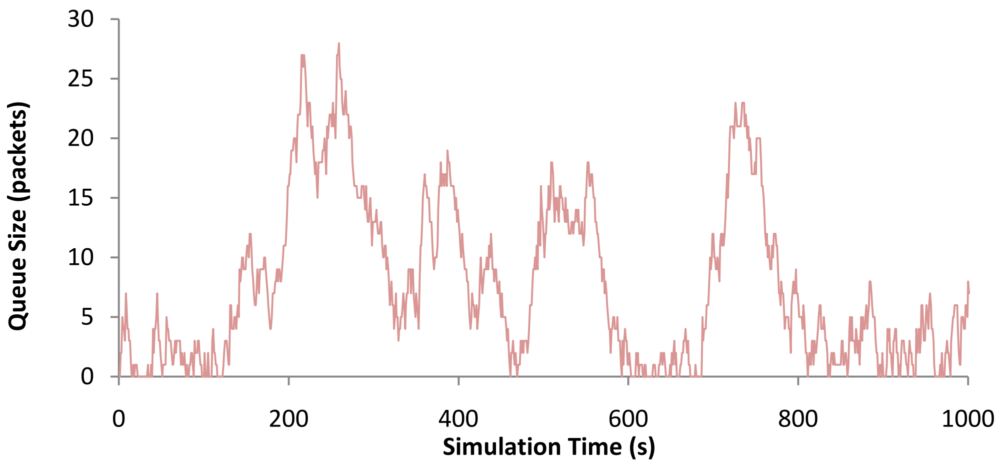


Figure 48. Actual queue size by the Poisson traffic. 10 second sample period, grey line: actual queue size.

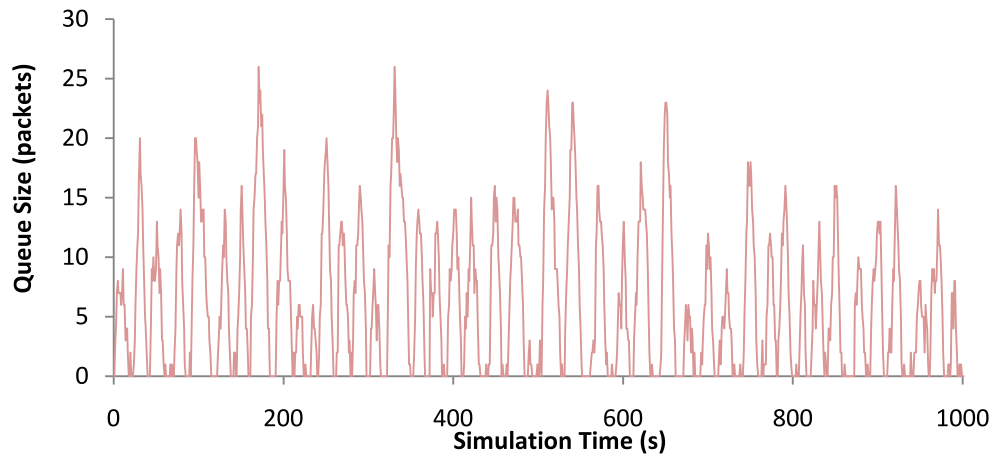


Figure 49. Actual queue size controlled by the controller model. Poisson traffic, 10 second sample period, grey line: actual queue size.

As shown in Figure 48 and Figure 49, the actual queue size distributed by Poisson traffic was controlled by the controller model with the nonlinear stochastic controller. However, this graph was not enough to compare the performance of the controller. A 30 run Monte Carlo analysis was conducted. The following graph (Figure 50) shows the result of a 30 run Monte Carlo analysis and the result by the OPNET.

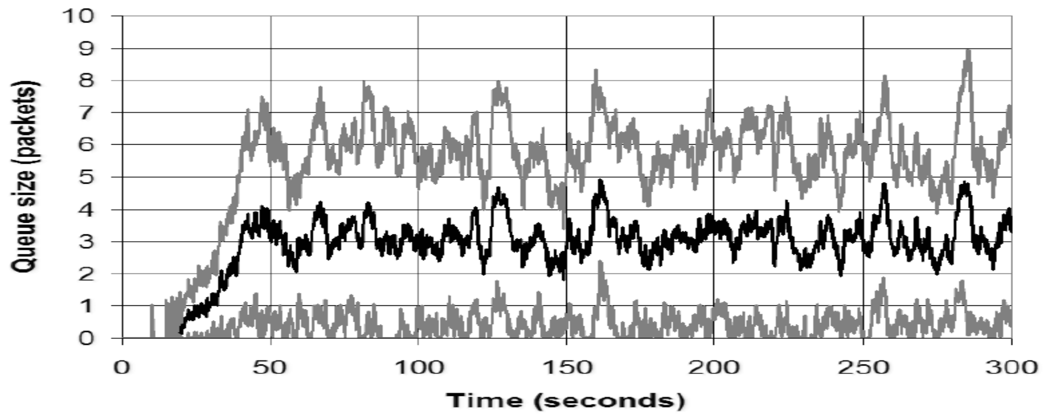


Figure 50. Simple network queue controller results by OPNET [15], yd = 3.125; X_{11} = 1, X_{22} = 1, X_{33}= 1, and U = 1000, 30-run Monte Carlo analysis, black line: mean, gray lines: mean plus or minus one standard deviation.

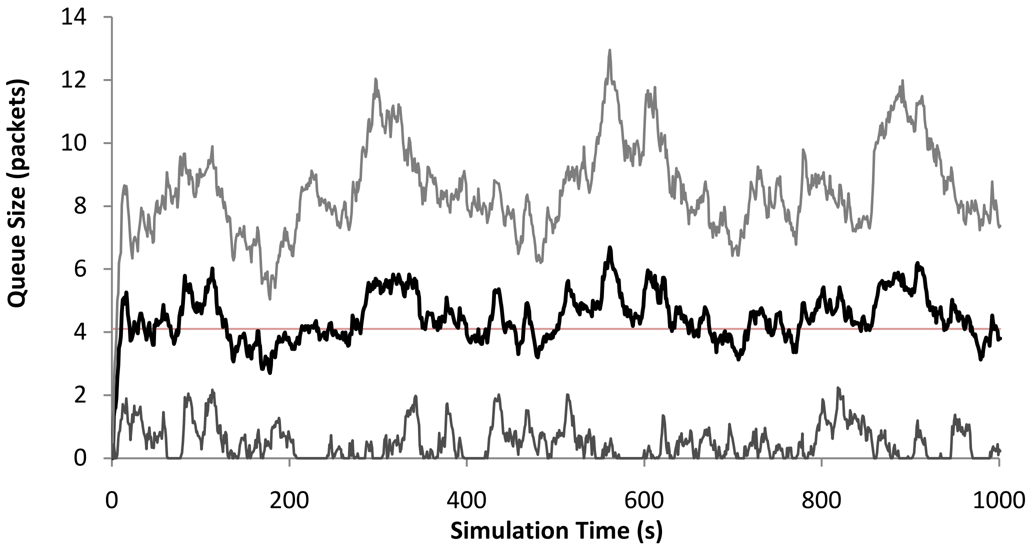


Figure 51. Simple network queue controller results by NS-2, yd = 3.125; X_{11} = 1, X_{22} = 1, X_{33}= 1, and U = 1000, 30-run Monte Carlo analysis, black line: mean, gray lines: mean plus or minus one standard deviation.

As shown in Figure 50 and Figure 51, the mean line was maintained following the setpoint y_d. However in NS-2 the mean of the queue size was 4.1 packets and this was more than the setpoint 3.125 packets. Moreover, the deviation of the NS-2 was more than the OPNET model. The standard deviation of the NS-2 model was maximum 6.34 packets while the OPNET model was maximum 4.18 packets. This was because of the difference between tools between NS-2 and OPNET model. Actually the constant value and output calculated by MATLAB are same with NS-2 and OPNET. Also because of the difference between NS-2 and OPNET, the tuning parameters were needed to be set again to maintain queue size accurately. The following graph (Figure 52) validates this statement.

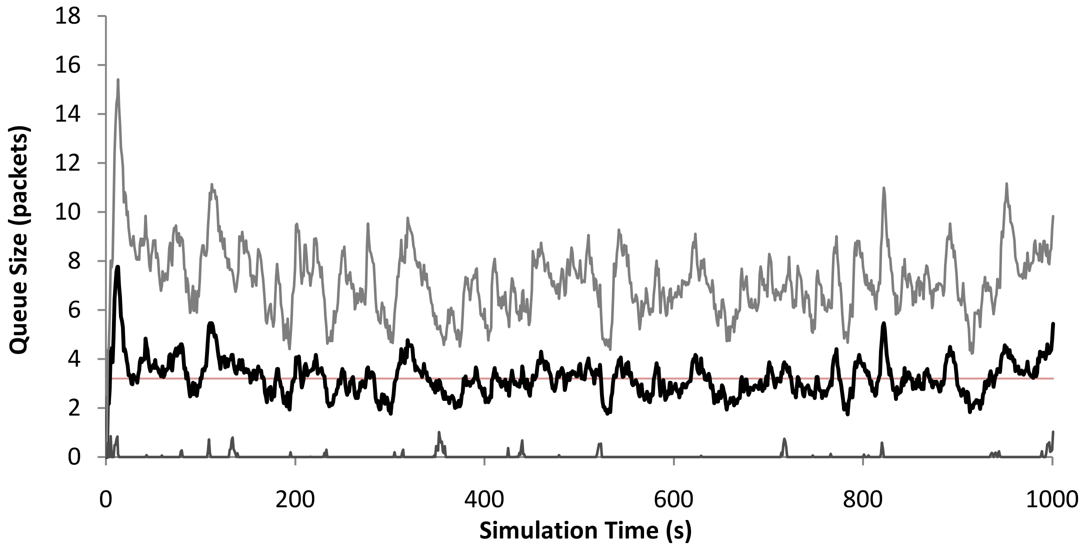


Figure 52. Simple network queue controller results by NS-2, yd = 3.125; X_{11} = 1, X_{22} = 100, X_{33}= 1, and U = 1000, 30-run Monte Carlo analysis, black line: mean, gray lines: mean plus or minus one standard deviation.

Figure 52 shows the result of changing the X_{22} with 100. This simulation was set with the same setpoint $y_d = 3.125$ and as shown in Figure 52, the mean of the queue size was 3.2 packets. The mean of the queue size in the OPNET model (Figure 50) was

2.97 packets. Therefore, the result of Figure 52 was good at controlling the queue size. However the difference between OPNET model and NS-2 model was that the deviation of the NS-2 model was more than the one of the OPNET model. And in the OPNET model, the result by changing X_{22} did not have the effect on the performance of the controller. But in the NS-2 model, when X_{22} was 100, the performance is better than X_{22} with one. This means that some tuning parameters need be set again in the NS-2 model and, although the controller in the NS-2 model had worse performance than the OPNET model, the NS-2 model was validated to regulate the queue size with the controller model designed by Stuckey's theory.

4.6. Summary

In this chapter, the transient queue model designed by Stuckey's theory was validated by the NS-2 model. This verified the work that ported the OPNET model into NS-2 with the MATLAB engine. With this validation, the network state estimator and controller were validated in the NS-2 model. In the network estimator model, it was validated in simple and mixed network model and the different traffic types. It presented that the performance differed by the traffic type, prediction time, packet size and traffic rate. In the network controller model, the performance of the network queue controller was investigated. Although the performance of the NS-2 model was lower than the OPNET model, it validated the Stuckey's theory to regulate the network queue size.

V. Conclusion

5.1. Contributions

In this thesis, Stuckey's stochastic algorithm with Kalman filter was ported into NS-2 successfully. Stochastic scheme is not developed largely; Stuckey developed the basic concepts and designs for stochastic estimation and control for network. Much of this document was devoted to verify the validity of this theory under sufficient experiments with NS-2.

The network models were designed from Stuckey's thesis to verify porting OPNET model into NS-2. This is represented in Section 3.5.3. These experiment scenarios are focus on porting into NS-2 but, also are focus on predicting network characteristics of queue size with different scenarios and traffic types. It was devoted to validate the Kalman filter by stochastic algorithm under various network situations.

In Section 4.2, the transient queue model was validated by NS-2. This was able to verify that the NS-2 model exactly performs the network experiment. This validates that the network model ported into NS-2 was verified Stuckey's model by the same network model. In the OPNET model, it performed the Kalman filter by calling the MATLAB engine. This also was ported into NS-2. When the Kalman filter predicted the queue size, the drop-tail queue in NS-2 was modified to call the MATLAB engine.

The performance of Kalman filter network estimator with NS-2 was shown in Section 4.3 and 4.4. In Section 4.3, the basic performance of network estimator is verified by the simple model. Also, we found the various relations under the different traffic types, packet sizes and traffic rates. Stuckey focused on the validation for Kalman filter estimator by the OPNET model but this thesis focused on the predictive ability of Kalman filter. This is shown by experiment for prediction time in Section 3.5.2. As a result, this thesis validates the performance and predicts actual queue size in the future. To measure this performance, we developed each simulation for traffic type, packet size, packet rate, and prediction time.

The best traffic type for performance is CBR and Pareto. CBR showed the best performance in MSE and Pareto is the best in MAPE (see Section 4.3.1). This result shows each performance for traffic type; especially, it shows that the network estimator can predict the actual queue size under 4 ~ 10 % relative error.

In simulation for traffic rate and packet size, the network estimator showed more efficiency when it predicted low traffic rate and high packet size (see Section 4.3.3 and 4.3.4). However, this is only in MSE with absolute error. In MAPE, it showed effect on the contrary. That is, although the changing rate and size influence queue size, this is only an effect about the actual queue, but is not for predictive performance. Therefore, the traffic rate and packet size do not have an effect on Kalman filter.

In simulation for prediction time, as expected, the performance of Kalman filter is better when prediction time is sooner. As described in Section 4.3.2, prediction time is same as the sample rate. The Kalman filter predicts queue size after sample rate with network information as sample rate. Accordingly, small sample rate increases network information, but decrease prediction time. It is obvious that a shorter prediction time shows better performance of the Kalman filter.

The network estimator was validated under ideal real network model. Various types of traffic were mixed as in a real network and under this network model, the performance of the Kalman filter estimator and the efficiency of placement for Kalman filter were verified. This validated the feasibleness of stochastic algorithm for real network.

In Section 4.5, the controller model was developed by the NS-2 model from the OPNET model to calculate the sending rate needed to regulate queue sizes. The controller model also had the tuning parameters (X_{11}, X_{22}, X_{33}, and U) and effect of these parameters was found by Stuckey's. The NS-2 controller model validated the performance of the controller with these parameters. It validated the difference effect from the OPNET model with X_{22}. Moreover, comparing the deviation of both NS-2 and OPNET model, the NS-2 model had lower performance in controller model. This was because of the difference of network simulation tools. However, this NS-2 model validate that the network controller can control sending rate to regulate the queue size and verify the Stuckey's theory. This also was devoted to validate that the controller model has the feasibleness for real network with stochastic algorithm.

5.2. Recommendations

This document comes from the foundation of Stuckey's thesis. As discussed in introduction, various media needs dynamic network. Accordingly, it is necessary that stochastic algorithm is adapted by solutions for real network. This document added validation about this feasibleness for future network.

One important characteristic is wireless for future network. The wired network cannot help having constraints for future network. Actually the dynamic network is based on wireless network. In this document, it is focus on porting into NS-2 and verifying Stuckey's theory, the main point that is dynamic network is ignored. Stochastic algorithm is needed to prepare for future network but a study is conducted under wired network. Consequently, stochastic algorithm should be conducted to study feasibleness for future network under wireless network.

This document focused on the experiment with the network simulation tool and, after verification with more tools, it was ported into NS-2. However, it was not enough to study the estimator and control model designed by Kalman filter. Most models in NS-2 were used to set the same configuration of the OPNET model. Equations that are used to develop Kalman filter in OPNET and NS-2 have improvements for initial error and error between predicted queue size and actual queue size. While not perfect in prediction, it will be able to decrease the error. Also, adjusting $Q_d, X_{11}, X_{22}, X_{33}$ and U in an iterative manner needs to be calculated to achieve the desired performance.

The network controller needs more testing in multiple network scenarios like real network. Investigation of the network estimator was conducted in the different traffic type and scenarios however, the network controller was not enough. To adjust the controller and to understand the effect by the parameters, multiple simulations were conducted. However, it was not validated with various scenarios. Especially, Scalability investigations should be conducted to study the feasibleness for real network.

The basic concept of stochastic algorithm already explored various fields. It was used in verifying the value in real world like GPS, Robot, Weather forecast, Network, etc. And it should be explored further. There are numerous network areas waiting to be explored with stochastic algorithms. In this document, the estimator with Kalman filter estimated the queue size and sending rate, but it could be developed to estimate network delay, waiting time, throughput, etc. This is a similar case as in the controllers with stochastic algorithms. For the future network, various stochastic algorithms will be developed real network applications.

Bibliography

1. Application of Kalman filter in WAAS
   http://www.gisdevelopment.net/technology/gps/techgp0033.htm

2. Attitude and Heading Reference Systems
   http://en.wikipedia.org/wiki/Attitude_and_Heading_Reference_Systems

3. Gilberto Flores Lucio, Marcos Paredes-Farrera, Emmanuel Jammeh, Martin Fleury, Martin J. Reed "OPNET Modeler and Ns-2: Comparing the Accuracy Of Network Simulators for Packet-Level Analysis using a Network Testbed" WSEAS Transactions on Computers, Vol. 2, July 2003

4. Greg Welch, Gary Bishop. "An Introduction to the Kalman Filter" University of North Carolina at Chapel *Hill*, TR 95-041, 24-July 2006.

5. Gross, D. and C. M. Harris. "Fundamentals of Queueing Theory (3rd Edition)", New York: Wiley-Interscience, 1998

6. Introduction to Monte Carlo simulation
   http://office.microsoft.com/en-us/excel/HA011118931033.aspx

7. James Trulove , "Broadband Networking", CRC Press, 2nd edition, 2000

8. Jonathan Pengelly "MONTE CARLO METHODS" University of Otago, Feb 2002

9. Kang, Sung-Joo, Won, You-Jip, Seong, Byeong-Chan "On-line Prediction Algorithm for Non-stationary VBR Traffic" Journal of KISS:Information Networking, 2007

10. MARTIN R. P., EDWARDS R. M. "Kalman filter application for distributed parameter estimation in reactor systems" Nuclear Science and Engineering, July 1996.

11. Mean squared error
    http://www.mste.uiuc.edu/patel/amar430/meansquare.html

12. Monte Carlo simulation basics
    http://www.vertex42.com/ExcelArticles/mc/MonteCarloSimulation.html

13. Monte Carlo simulation in MS Excel
    http://www.projectsmart.co.uk/docs/monte-carlo-simulation.pdf

14. NAM: Network Animator
http://isi.edu/nsnam/nam/

15. Nathan C. Stuckey. "STOCHASTIC ESTIMATION AND CONTROL OF QUEUES WITHIN A COMPUTER NETWORK" MS Thesis. AFIT/GE/ENG/07-24. Air Force Institute of Technology, 22-Mar 2007.

16. NAVSTAR GPS USER EQUIPMENT INTRODUCTION, Sep, 1996 http://www.navcen.uscg.gov/pubs/gps/gpsuser/gpsuser.pdf

17. OPNET website
www.opnet.com

18. Poisson traffic generator for ns-2
http://ipv6.willab.fi/kostas/src/Application-Traffic-Poisson/

19. Rick Middleton. "Applications of the Kalman Filter Algorithm to Robot Localization and World Modelling" Electrical Engineering Final Year Project 2002, University of Newcastle, 2002.

20. Rui Castro, Mark Coates, Gang Liang, Robert Nowak, Bin Yu. "Network Tomography: recent developments" National Science Foundation. 7-Mar 2003.

21. Sensor Fusion
http://www.u-dynamics.com/sensor_fusion/sensfuse-datasheet.pdf

22. Seong-Whan Kim, Ki-Hong Ko "Kalman Filter Based Dead Reckoning Algorithm for Minimizing Network Traffic Between Mobile Game Users in Wireless GRID" Springer Berlin / Heidelberg, Feb 20, 2008

23. Song, Ci, et al. .A Link Adaptation Approach for QoS Enhancement in Wireless Networks,.IEEE Conference on Local Computer Networks (LCN), 2001

24. Stefanie Jachner, K. Gerald van den Boogaart, Thomas Petzoldt "Statistical Methods for the Qualitative Assessment of Dynamic Models with Time Delay" journal of statistical software, Sep 2007

25. The NS manual (formerly *ns* Notes and Documentation), June 19, 2008 http://isi.edu/nsnam/ns/doc/everything.html

26. The Otcl Tutorial, 1995.
http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/tutorial.html

27. Thomas g. Robertazzi. "Computer networks and systems: queueing theory and performance evaluation" Springer, June 22, 200

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. REPORT DATE *(DD-MM-YYYY)* 26-03-2009	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* Sep 2008 – Mar 2009

4. TITLE AND SUBTITLE STOCHASTIC ESTIMATION AND CONTROL OF QUEUES WITHIN A COMPUTER NETWORK	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Kim, Mingook, Capt, ROKA	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCS/ENG/09-04
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Captain Nathan C. Stuckey implemented the idea of the stochastic estimation and control for network in OPNET simulator. He used extended Kalman filter to estimate packet size and packet arrival rate of network queue to regulate queue size. To validate stochastic theory, network estimator and controller is designed by OPNET model. These models validated the transient queue behavior in OPNET and work of Kalman filter by predicting the queue size and arrival rate. However, it was not enough to verify a theory by experiment. So, it needed to validate the stochastic control theory with other tools to get high validity. Our goal was to make a new model to validate Stuckey's simulation. For this validation, NS-2 was studied and modified the Kalman filter to cooperate with MATLAB. Moreover, NS-2 model was designed to predict network characteristics of queue size with different scenarios and traffic types. Through these NS-2 models, the performance of the network state estimator and network queue controller was investigated and shown to provide high validity for Stuckey's simulations.

15. SUBJECT TERMS
Kalman filter, Stochastic algorithm, NS-2, OPNET, MATLAB, Queue, network state estimator, network queue controller

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Lt Col. Stuart H. Kurkowski
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	98	**19b. TELEPHONE NUMBER** *(Include area code)* 937-785-3636 x7228 stuart.kurkowski@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18